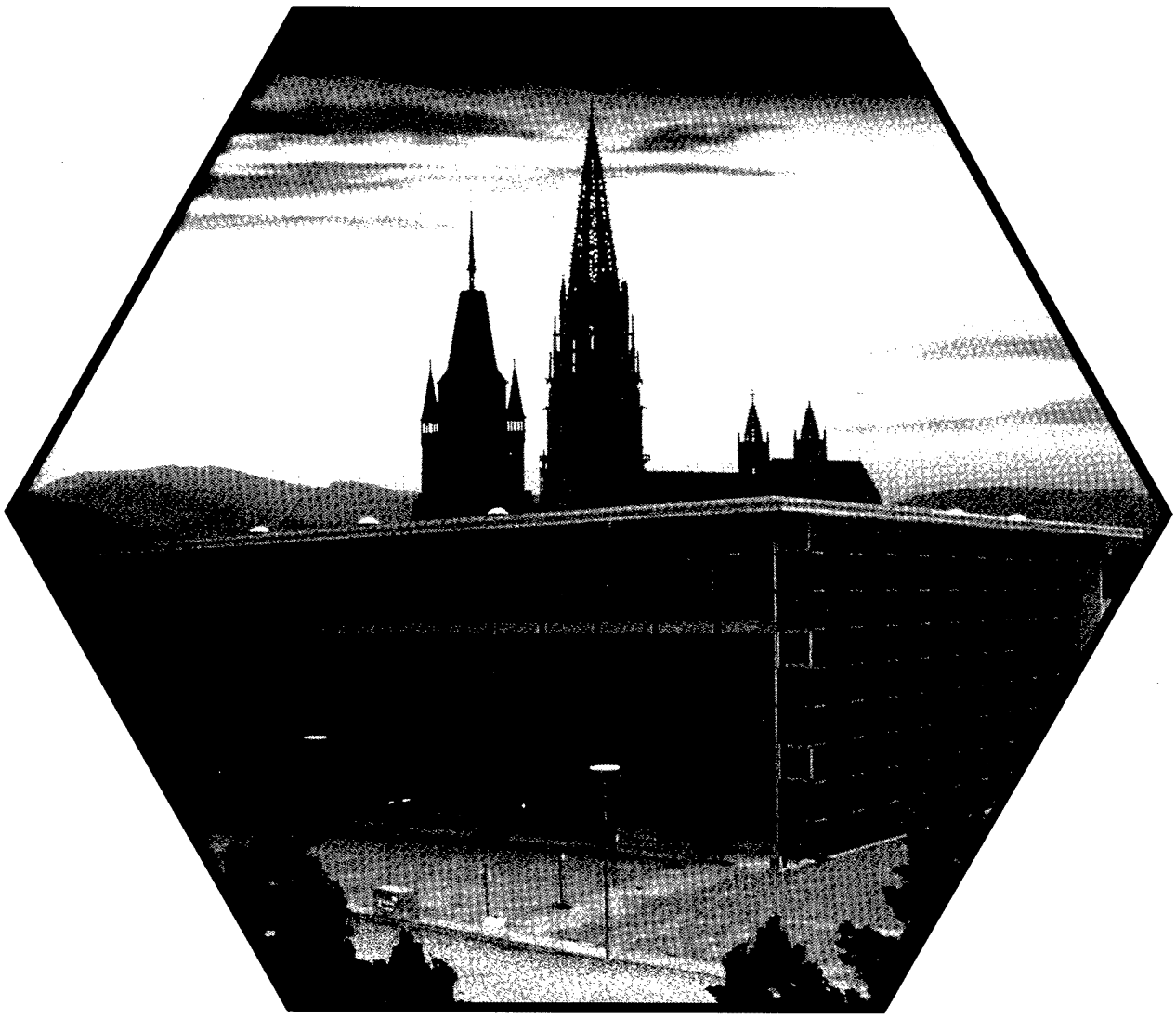
PROCEEDINGS

THE TWENTY-NINTH IEEE INTERNATIONAL SYMPOSIUM ON MULTIPLE-VALUED LOGIC



MAY 20–22, 1999
FREIBURG, GERMANY

Sponsored by
IEEE Computer Society Technical Committee on Multiple-Valued Logic
Albert-Ludwigs-University Freiburg, Germany

DTIC QUALITY INSPECTED 4

19990719 060

IEEE COMPUTER SOCIETY

DISTRIBUTION STATEMENT A
Approved for Public Release
Distribution Unlimited

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	8 July 1999	Conference Proceedings

4. TITLE AND SUBTITLE

29th International Symposium on Multiple-Valued Logic 1999

5. FUNDING NUMBERS

F61775-99-WF053

6. AUTHOR(S)

Conference Committee

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Albert Ludwigs University Freiburg
Am Flughafen 17
Freiburg 79110
Germany

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

EOARD
PSC 802 BOX 14
FPO 09499-0200

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

CSP 99-5053

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

A

13. ABSTRACT (Maximum 200 words)

The Final Proceedings for 29th International Symposium on Multiple-Valued Logic 1999, 20 May 1999 - 22 May 1999

This is an interdisciplinary conference consisting of topics in: high speed computation, optical computing, fault detection and diagnosis, logic design and switching theory, decision diagrams, programmable logic, circuit/device implementation, reliability, algebraic and formal aspects, fuzzy logic, artificial intelligent systems, philosophical aspects, and automated deduction.

14. SUBJECT TERMS

EOARD, Advanced Computing, Mathematics, Fuzzy Logic

15. NUMBER OF PAGES

302

16. PRICE CODE

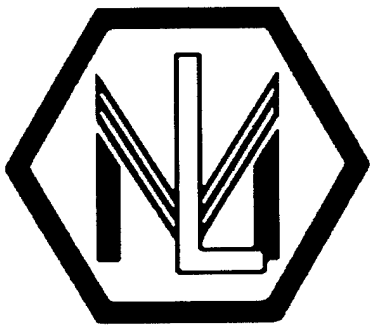
N/A

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

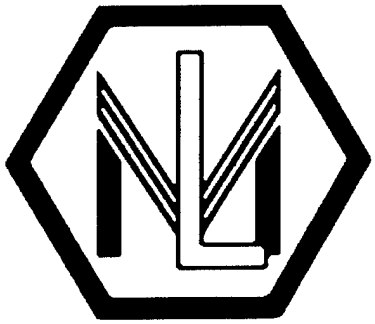
PROCEEDINGS

1999

29TH IEEE INTERNATIONAL SYMPOSIUM ON MULTIPLE-VALUED LOGIC



AQF99-10-1834



PROCEEDINGS

1999

29TH IEEE INTERNATIONAL SYMPOSIUM ON MULTIPLE-VALUED LOGIC

May 20 – 22, 1999

Freiburg im Breisgau, Germany

Sponsored by

IEEE Computer Society Technical Committee on Multiple-Valued Logic

Albert-Ludwigs-University • Freiburg im Breisgau, Germany

IEEE COMPUTER SOCIETY

Los Alamitos, California

Washington • Brussels • Tokyo

Copyright © 1999 by The Institute of Electrical and Electronics Engineers, Inc.
All rights reserved

Copyright and Reprint Permissions: Abstracting is permitted with credit to the source. Libraries may photocopy beyond the limits of US copyright law, for private use of patrons, those articles in this volume that carry a code at the bottom of the first page, provided that the per-copy fee indicated in the code is paid through the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923.

Other copying, reprint, or republication requests should be addressed to: IEEE Copyrights Manager, IEEE Service Center, 445 Hoes Lane, P.O. Box 133, Piscataway, NJ 08855-1331.

The papers in this book comprise the proceedings of the meeting mentioned on the cover and title page. They reflect the authors' opinions and, in the interests of timely dissemination, are published as presented and without change. Their inclusion in this publication does not necessarily constitute endorsement by the editors, the IEEE Computer Society, or the Institute of Electrical and Electronics Engineers, Inc.

IEEE Computer Society Order Number PR00161
ISBN 0-7695-0161-3
ISBN 0-7695-0163-X (microfiche)
ISSN 0195-623X
IEEE Order Plan Catalog Number 99CB36329

Additional copies may be ordered from:

IEEE Computer Society
Customer Service Center
10662 Los Vaqueros Circle
P.O. Box 3014
Los Alamitos, CA 90720-1314
Tel: + 1-714-821-8380
Fax: + 1-714-821-4641
cs.books@computer.org

IEEE Service Center
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
Tel: + 1-732-981-0060
Fax: + 1-732-981-9667
mis.custserv@computer.org

IEEE Computer Society
Watanabe Building
1-4-2 Minami-Aoyama
Minato-ku, Tokyo 107-0062
JAPAN
Tel: + 81-3-3408-3118
Fax: + 81-3-3408-3553
tokyo.ofc@computer.org

Editorial production by Thomas Baldwin

Cover photograph by Andreas Hett

Cover art production by Alex Torres

Printed in the United States of America by Technical Communications Services

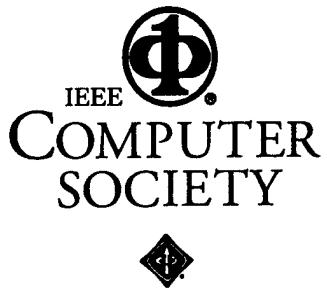


TABLE OF CONTENTS

TWENTY NINTH IEEE INTERNATIONAL SYMPOSIUM ON MULTI-VALUED LOGIC (ISMVL'99)

Message from the Symposium Chairs viii
Message from the Program Co-Chairs ix
Symposium Committee x
Referees xi

• SESSION 1 — INVITED ADDRESS •

Development of Quantum Functional Devices for Multiple-Valued Logic Circuits 2
T. Baba

• SESSION 2A — ALGEBRA I •

Multivalued Binary Relations and Post Algebras 10
M. Serfati

Quaternion Groups versus Dyadic Groups in Representations and Processing of Switching Functions 18
R. S. Stanković, D. Milenović, D. Janković

On Axiomatization of Conditional Entropy of Functions Between Finite Sets 24
S. Jaroszewicz, D. Simovici

• SESSION 2B — CIRCUITS I •

Multiple-Valued Content-Addressable Memory Using Metal-Ferroelectric-Semiconductor FETs 30
T. Hanyu, H. Kimura, M. Kameyama

"New lamps for old!" (Generalized Multiple-valued Neurons) 36
C. Moraga, R. Heider

Supplementary Symmetrical Logic Circuit Structure 42
E. Olson

• SESSION 3A — DECOMPOSITION •

Bi-Decompositions of Multi-Valued Functions for Circuit Design and Data Mining Applications 50
B. Steinbach, M. Perkowski, C. Lang

Totally Undecomposable Functions: Applications to Efficient Multiple-Valued Decompositions 59
T. Sasao

A Generalization of Shestakov's Function Decomposition Method 66
J. J. Lou, J. Brzozowski

• SESSION 3B — CLONES •

Gigantic Pairs of Minimal Clones 74
I. G. Rosenberg, H. Machida

Maximal Chains of Partial Clones Containing All Idempotent Partial Functions 80
L. Haddad, J. Fugère

Partial Clones and their Generating Sets 85
L. Haddad, D. Lau

• SESSION 4A — LOGIC DESIGN •

Evaluation of m-valued Fixed Polarity Generalizations of Reed-Muller Canonical Form 92
E. Dubrova

Multiple-Valued Minimization to Optimize PLAs with Output EXOR Gates 99
D. Debnath, T. Sasao

The Output Permutation for the Multiple-Valued Logic Minimization with Universal Literals 105
T. Hozumi, O. Kakusho, Y. Hata

Logical Model for Representing Uncertain Statuses of Multiple-Valued Logic Systems Realized by Min, Max and Literals 110
N. Takagi, A. Hon-nami, K. Nakashima

• SESSION 4B — ALGEBRA II •

A Super Switch Algebra for Quantum Device Based Systems 118
G. Dueck, M. Hu, B. Fraser

Clarifying the Axioms of Kleene Algebra based on the Method of Indeterminate Coefficients 125
T. Ninomiya, M. Mukaidono

The Number of Cascade Functions 131
G. Pogosyan

Research on the Similarity among Precomplete Sets Preserving m-ary Relations in Partial K-Valued Logic 136
R. Liu

• SESSION 5 — INVITED ADDRESS •

Structural and Behavioral Modeling with Monadic Logics 142
A. Ayari, D. Basin, S. Friedrich

• SESSION 6A — DECISION DIAGRAMS •

Matrix-Valued EXOR-TDDs in Decompositon of Switching Functions 154
R. Stanković

Synthesis of Multiple-Valued Decision Diagrams using Current-Mode CMOS Circuits 160
M. Abd-El-Barr, H. Fernandes

Shared Multiple-Valued Decision Diagrams for Multiple-Output Functions 166
H. Md. H. Babu, T. Sasao

• SESSION 6B — CIRCUITS II •

Ternary Multiplication Circuits Using 4-Input Adder Cells and Carry Look-Ahead 174
A. Herrfeld, S. Hentschke

Down Literal Circuit with Neuron-MOS Transistors and Its Applications 180
J. Shen, K. Tanno, O. Ishizuka

Arithmetic Circuits for Analog Digits 186
A. Saed, M. Ahmadi, G. A. Jullien

• SESSION 7A — APPLICATIONS •

Quaternary Coded Genetic Algorithms 194
K. Freitag, L. Hildebrand, C. Moraga

Redundant Complex Arithmetic and Its Application to Complex Multiplier Design 200
T. Aoki, K. Hoshi, T. Higuchi

On the Number of Multilinear Partitions and the Computing Capacity of Multiple-Valued Multiple-Threshold Perceptrons 208
A. Ngom, I. Stojmenović, J. Žunić

B-ternary Logic Based Asynchronous Micropipeline 214
Y. Nagata, D. M. Miller, M. Mukaidono

State Assignment Techniques in Multiple-Valued Logic 220
K. Adams, J. Campbell, L. Maguire, J. Webb

• SESSION 7B — LOGIC •

Information Relationships and Measures in Application to Logic Design 228
L. Jóźwiak

Probabilistic and Truth-Functional Many-Valued Logic Programming 236
T. Lukasiewicz

Representation Theorems and Theorem Proving in Non-Classical Logics 242
V. Sofronie-Stokkermans

Transformations between Signed and Classical Clause Logic 248
B. Beckert, R. Hähnle, F. Manyà

Semirigid Problems in k-Valued Logic 256
M. Miyakawa

• SESSION 8 — PANEL SESSION •

Multiple-Valued Logic in the Next Millenium: Challenges and Perspectives

• SESSION 9A — TESTING •

Fault Characterization and Testability Considerations in Multi-Valued Logic Circuits 262
M. Abd-El-Barr, M. Al-Sherif, M. Osman

Highly Testable Boolean Ring Logic Circuits 268
U. Kalay, M. Perkowski, D. Hall

Self-Checking Multiple-Valued Circuit Based on Dual-Rail Current-Mode Differential Logic 275
T. Hanyu, T. Ike, M. Kameyama

• SESSION 9B — FUZZY LOGIC •

On the Concept of Qualitative Fuzzy Set 282
H. Thiele

On Some Classes of Fuzzy Information Granularity and Their Representations 288
Y. Hata, M. Mukaidono

From a Fuzzy Flip-Flop to a MVL Flip-Flop 294
L.P. Maguire, T.M. McGinnity, L.J. McDaid

Author Index 302

MESSAGE FROM THE SYMPOSIUM CHAIRS

Welcome to Freiburg im Breisgau, the site of the 29th International Symposium on Multiple-Valued Logic. Freiburg is a period middle-ages city founded by the Dukes of Zähringen (the right to hold markets was granted in 1120 A.D.). Its historical development was especially influenced by its more than 400-year-long association with the Habsburg Dynasty. The city developed from a walled middle-ages market town to a fort city (17th/18th century) to the present-day modern city of about 200,000 inhabitants. It is the first time that the symposium is being held in Germany.

Like in the past years researchers from different areas discuss the latest results in the field of multiple-valued logic. The symposium is co-sponsored by the Albert-Ludwigs-University, Freiburg, and the IEEE Computer Society.

We wish to express our gratitude to the Symposium Committee for their hard work in preparing this event. Dr. Elena Dubrova served as a Program Chair for Europe and Africa, Dr. Takahiro Hanyu served as a Program Chair for Asia and Australia, and Prof. Michael Miller served as a Program Chair for the Americas. We would like to thank them for organizing an excellent program. ISMVL'99 has been the joint effort of many people. We especially like to thank Nicole Drechsler (Financial Chair), Frank Schmiedle (Publication Chair), and Dr. Christoph Scholl (Local Arrangement Chair).

We would also like to thank Thomas Baldwin of the IEEE Computer Society for his work in publishing this volume.

Finally, we would like to wish all participants a beautiful time in Freiburg and the Black Forest area and hope that we will have many stimulating discussions.

Rolf Drechsler
Symposium Chair

Bernd Becker
Symposium Co-Chair

MESSAGE FROM THE PROGRAM CO-CHAIRS

We are pleased to extend our welcome to the 1999 International Symposium on Multiple-Valued Logic. It has been our pleasure to have coordinated the selection of a technical program which we are confident you will find interesting and stimulating. This year there are 42 contributed papers presented by MVL researchers from around the world. There are sessions on algebra, applications, clones, circuits, decision diagrams, decomposition, fuzzy logic, logic, logic design, and testing. This year's program is thus, as in the past, representative of the diverse interest in multiple-valued logic research across many disciplines.

This year's program includes two invited addresses. Dr. Toshio Baba, NEC Japan will address *Development of Quantum Functional Devices for Multiple-Valued Logic*. Prof. Dr. David Basin, University of Freiburg, will speak on *Structural and Behavioral Modeling with Monadic Logics*. We thank each of the invited speakers for agreeing to address the symposium and for their written contributions which appear in these proceedings.

It is fitting as we rapidly approach the transition to the year 2000, that this year's symposium will have a panel session on *Multiple-Valued Logic in the Next Millenium: Challenges and Perspectives*. We look forward to the views of the panelists and indeed all participants at ISMVL-99.

For each ISMVL, contributed papers go through a rigorous reviewing process based on full paper submissions. We thank all authors who contributed their work for consideration for presentation at this year's symposium. On our own behalf, and on behalf of all the authors, we also thank the referees, listed elsewhere in these proceedings, who contributed their time and expertise to the selection process. Their efforts are crucial to the success of a meeting such as ISMVL.

On behalf of all attendees, we wish to thank Dr. Rolf Drechsler, Symposium Chair and Prof. Bernd Becker, Symposium Co-Chair for their extensive efforts in the organization of this year's ISMVL. We also thank Nicole Drechsler, Christoph Scholl and Frank Schmiedle for their contributions to the Symposium. And finally we thank the staff at the IEEE Computer Society, for their efforts in preparing these proceedings.

We hope that you truly enjoy all aspects of ISMVL'99 and that you will feel encouraged to participate in ISMVL'2000 and future symposia.

Elena Dubrova (Europe-Africa)
Takahiro Hanyu (Asia-Pacific)
D. Michael Miller (Americas)

ISMVL-99 Program Co-Chairs

Symposium Committee

Symposium Chair
Rolf Drechsler, *University of Freiburg*

Symposium Co-Chair
Bernd Becker, *University of Freiburg*

Financial Chair
Nicole Drechsler, *University of Freiburg*

Local Arrangement Chair
Christoph Scholl, *University of Freiburg*

Publication Chair
Frank Schmiedle, *University of Freiburg*

Program Committee

Americas Co-Chair
Michael Miller, *University of Victoria*

Asia/Pacific Co-Chair
Takahiro Hanyu, *Tohoku University*

Europe/Africa Co-Chair
Elena Dubrova, *Royal Institute of Technology, Kista*

Session Chairs

Session1: Takahiro Hanyu
Session2A: Ivo Rosenberg
Session2B: Yutaka Hata
Session3A: Jon Muzio
Session3B: Grant Pogosyan
Session4A: Claudio Moraga
Session4B: Reiner Hähnle
Session5: Elena Dubrova
Session6A: Rolf Drechsler
Session6B: Michitaka Kameyama
Session7A: Radomir Stanković
Session7B: Lucien Haddad
Session8: Michael Miller
Session9A: Tsutomu Sasao
Session9B: Okihiko Ishizuka

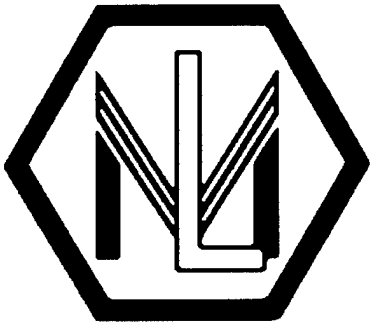
Referees

Mostafa Abd-El-Barr
Seiki Akama
Takafumi Aoki
Tomoyuki Araki
Matthias Baaz
Hafiz Md. Hasan Babu
Bernhard Beckert
Robert J. Bignall
Ferdinand Börner
Jon T. Butler
K. W. Current
Debatosh Debnath
S. Demri
Rolf Drechsler
Elena Dubrova
Gerhard W. Dueck
Daniel Etiemble
Bogdan J. Falkowski
Craig M. Files
Glen Gulak
Dilian Gurov
Lucien Haddad
Reiner Hähnle
Takahiro Hanyu
Yutaka Hata
Andreas Herrfeld
Takahiro Hozumi
Mou Hu
Okihiko Ishizuka
Tatiana Kalganova
Michitaka Kameyama
Naotake Kamiura
Hiroaki Kikuchi
Tadahiro Kitahashi
Eiji Kobayashi
Koji Kotani
Thomas Lukasiewicz
Luca Macchiarulo
Hajime Machida
Felip Manyà
Masayuki Matsumoto
D. Michael Miller
Shin-ichi Minato
Masahiro Miyakawa
Claudio Moraga
Masao Mukaidono
Noriaki Muranaka
Jon C. Muzio
Yasunori Nagata
Akira Nakamura
Alioune Ngom
A. Nozaki
Manuel Ojeda-Aciego
Ewa Orlowska
Marek Perkowski
Grant R. Pogosyan
Ivo Rosenberg
C. Rozon
Gernot Salzer
Tsutomu Sasao
Hiroshi Sawada
Micaela Serra
Charles B. Silio
Dan Simovici
Katsuhiko Shimabukuro
Vlad Petrovich Shmerko
Viorica Sofronie-Stokkermans
Radomir S. Stanković
Bernhard von Stengel
Ivan Stojmenović
Iwan G. Tabakow
Noboru Takagi
K. Tanno
Hisayuki Tatsumi
Mitchell Thornton
Zvonko Vranesic
Shugang Wei
David M. Wessels
Yasushi Yuminaka
Richard Zach
Zelko Zilic

SESSION I

INVITED ADDRESS

Chair: Takahiro Hanyu



Development of Quantum Functional Devices for Multiple-Valued Logic Circuits

Toshio Baba
Fundamental Research Laboratories, NEC Corporation
34, Miyukigaoka, Tsukuba, Ibaraki 305-8501, Japan
baba@frl.cl.nec.co.jp

Abstract

Quantum functional devices exhibiting unique current-voltage characteristics are reported for the application of multiple-valued logic circuits. Multiple negative-differential-resistance (NDR) characteristics in drain current-voltage characteristics are demonstrated by using multiple-junction surface tunnel transistors (MJ-STTs). Some multiple-valued logic gates such as inverter and literal are implemented using the MJ-STTs. Oscillatory characteristics of drain current under gate modulation are shown by single electron transistors. Nonvolatile multiple-valued memory devices utilizing these unique characteristics are described, and the fundamental operation of write and read of stored electrons are demonstrated.

1. Introduction

The operation speed and memory density of ultra large-scale integrated circuits (ULSI) is increasing dramatically because the device size has been decreased according to the scaling rule. Recently, however, severe problems with regard to complex wiring, large propagation delay time, and high power consumption are being discussed in order to improve the performance of ULSIs. Even for MOSFETs, which is key component in ULSIs, enhancement of speed through size-reduction tends to be restricted because of an increase in short channel effects and gate leakage current. Therefore, any new technology from circuit architecture to transistor is strongly desired to overcome these problems and to improve the performance of ULSIs in the future.

The quantum functional devices (QFD) project in Japan began in 1991 in order to develop new devices that utilize quantum effects for overcoming the problems on future ULSIs [1]. This project lasts ten years and is divided into three terms (Fig. 1). Beginning with Phase I, which consists of the development from basic technology for new devices, to Phase III, which consists of integrated-circuit technology, devices and circuits that are promising for future ULSIs are being developed at a level close to practical use. New quantum functional devices are proposed by the following eight member companies: NEC, Fujitsu, Motorola, Matsushita Electric, Sony, Hitachi, Hitachi Europe, and Toshiba. These devices have unique current-voltage characteristics compared with the conventional MOSFETs and bipolar transistors. These devices are thought to be very useful for binary logic and/or memories because they can reduce the number of components and wires, and they can lower the power consumption. In addition, some devices could be applicable to multiple-valued logic circuits, which may further reduce the circuit components than those for the binary circuits.

Negative-differential-resistance (NDR) characteristics, which appear in most of the quantum functional devices, are key features that reduce the circuit components. Several new binary logic circuits have been constructed using resonant tunneling transistors (RTT), which include resonant tunneling diode (RTD) structures [2, 3]. Furthermore, multiple NDR characteristics are very promising for the application of multiple-valued logic circuits because of clear multiple threshold characteristics. These characteristics were also developed by series connections of RTDs or RTTs [4, 5].

We proposed the surface tunnel transistor (STT) as one of the quantum functional devices because it exhibits the NDR characteristics in the drain current-voltage characteristics mentioned under the QFD project [6]. Since this device has a planar structure similar to the MOSFET, integration of devices is much easier to the vertical structure devices like RTDs. Because of this structural feature, multiple NDR characteristics are demonstrated using a multiple-junction surface tunnel transistor (MJ-STT), which consists of series connected tunnel-junctions between a source and a drain [7, 8].

Multiple NDR characteristics are also seen in the drain

0-7695-0161-3/99 $10.00 © 1999 IEEE

91	92	93	94	95	96	97	98	99	2000
Phase I Concept design of QFD and basic technologies for fabrication				Phase II Development and estimation for element QFD and functional block			Phase III Development of advanced and integrated QFDs		

Figure 1. Quantum functional device (QFD) project's long term research plan.

current as a function of the gate voltage for single electron transistors (SETs) [9]. This unique feature is also attractive for the multiple-valued logic circuits. Because the SETs are inherently very tiny structures with nano-meter scale and because they can operate with very low power consumption, very large scale integration are expected.

In this paper, some multiple-valued logic and memory circuits using MJ-STTs and SETs are proposed from NEC as examples of the application for multiple-valued logic circuits using the QFDs. Prospects of multiple-valued logic circuits using the QFDs are also discussed.

2. Multiple-valued logic circuits using a multiple-junction surface tunnel transistor

Figure 2 shows a schematic cross-section of the multiple-junction surface tunnel transistor (MJ-STT) and its expected transistor characteristics with corresponding band diagrams. In this structure, the isolated p^+ regions (p^+ islands) are inserted in the n^+ channel between the p^+ source and the p^+ drain. Many p^+/n^+ interband tunnel junctions connected in series are formed so that they correspond to the number of p^+ islands. Because each forward-biased p^+/n^+ junction generates NDR characteristics, multiple NDR characteristics are expected for MJ-STTs..

InGaAs-based MJ-STTs were fabricated using molecular-beam-epitaxy (MBE) regrowth [7]. After the formation of the isolated p^+-region, the n^+ -channel and gate layers are deposited on this structure by MBE regrowth. A quasi planar device structure is formed using this fabrication process. Details of the fabrication process and the parameters of the devices are written in reference 8.

An example of the transistor characteristics for MJ-STTs is shown in Fig. 3. Transistor characteristics with six NDRs were clearly observed for the fabricated MJ-STT with 6 forward-biased tunnel junctions. The exact relation between the numbers of tunnel junctions and NDRs were confirmed by fabricating the MJ-STTs with different

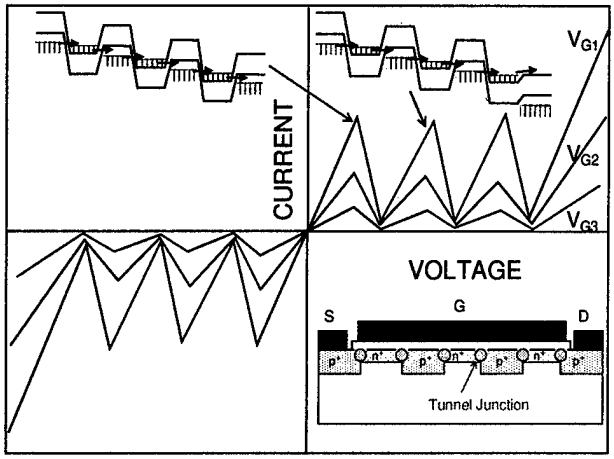


Figure 2. Expected transistor characteristics of the multiple-junction surface tunnel transistor (MJ-STT). Insets show the schematic of an ideal MJ-STT structure and the band diagrams between the source and the drain.

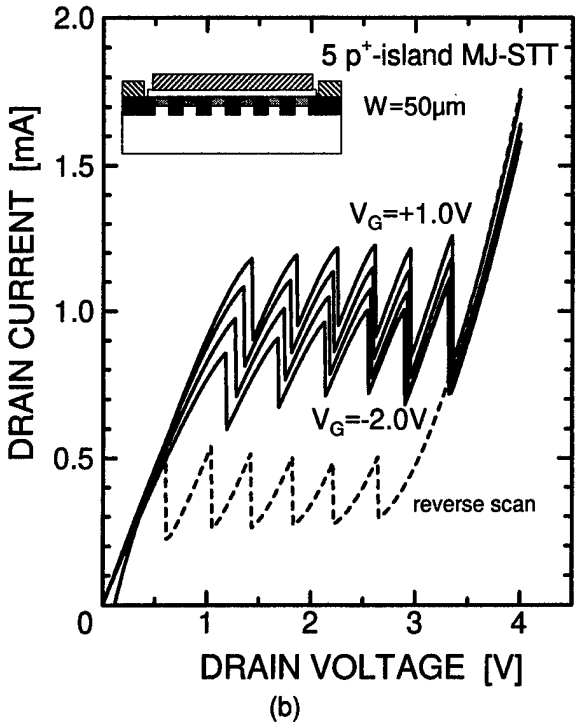


Figure 3. Transistor characteristics of a fabricated InGaAs MJ-STT with 6 forward-biased tunnel junctions. The broken-line curve represents the reverse scan at V_G = 0V.

numbers of tunnel junctions.

Two types of fundamental multiple-valued logic circuits were designed for MJ-STTs and they were implemented monolithically using the InGaAs-based MJ-STTs.

Figure 4(a) is a schematic cross-section, 4(b) is a microscope photograph, and 4(c) is the equivalent circuit for a fabricated three-valued inverter circuit. This circuit consists of series-connected one-p^+-island MJ-STTs. The operating principle of the fabricated logic circuits is based on either monostable-bistable transition logic (MOBIL) or monostable-multistable transition logic (MML) [10, 5].

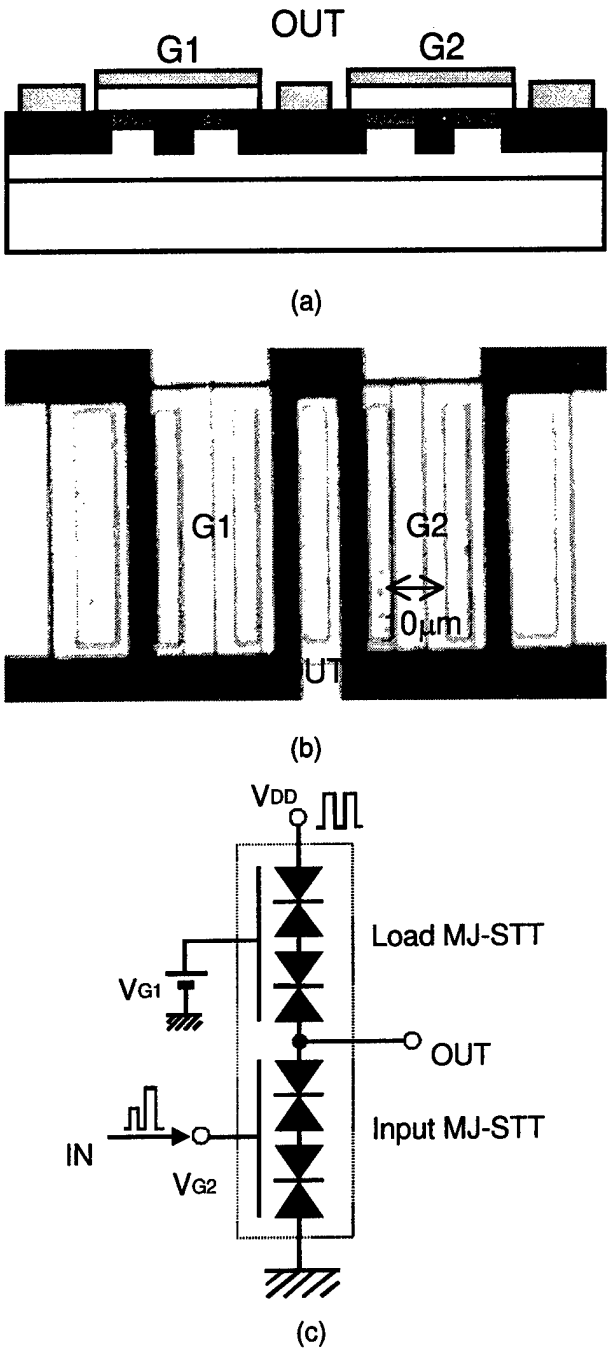


Figure 4. Three-valued inverter element: (a) Schematic cross-section, (b) microscope photograph and (c) equivalent circuit.

Figure 5 shows the input and output traces of the three-valued inverter circuit depicted in Fig. 4. Based on the results of measurements for the fabricated device, we chose a clocked supply voltage of 1.75 V. The input voltages were –1.4 V for the "0" logic level, -0.6 V for the "1" logic level, and 0.2 V for the "2" logic level. The operation of the three-valued inverter can be clearly seen in Fig. 5. Furthermore, it is found that the output is kept even after the input pulse returns to zero. This indicates that this inverter circuit has a latch function.

Figure 6(a) is a schematic cross-section, 6(b) is a microscope photograph, and 6(c) is the equivalent circuit of a fabricated literal circuit. This circuit consists of series-connected zero-p^+-island MJ-STTs and parallel-connected zero-p^+-island MJ-STTs. The operation of this circuit is also based on the MML using clocked supply voltage. To explain the operation principle of this literal circuit, which was originally proposed by Waho using RTDs, Fig. 7 shows the peak current for each MJ-STT in this circuit as a function of the input voltage [5]. Each MJ-STT is labeled to "X1", "X2", "X3", and "X4". Due to the feature of the MML or series connected NDR elements, a device having the lowest peak current goes to a high voltage state first when the supply voltage is raised from 0 V . Based on this principle, the output voltage increases only when the input voltage is between V_A and V_B, where the sum of the peak current for X3 and X4 is lower then those for X1 and X2. Therefore, literal operation can be achieved in this circuit.

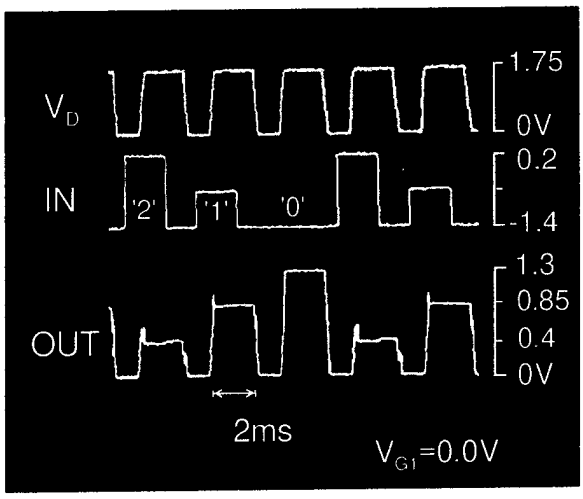
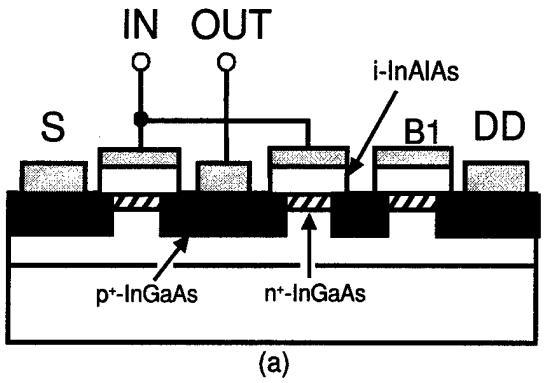
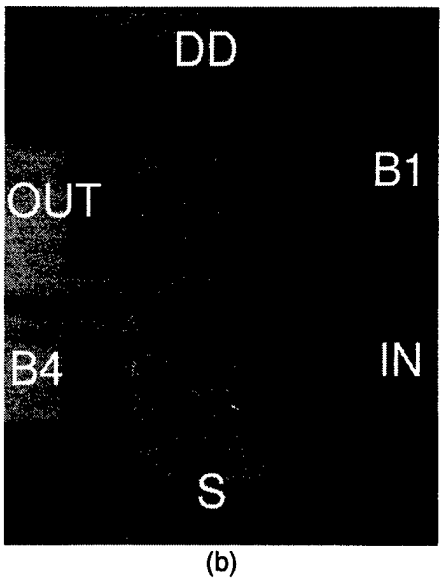


Figure 5. Input and output traces of the three-valued inverter circuit.





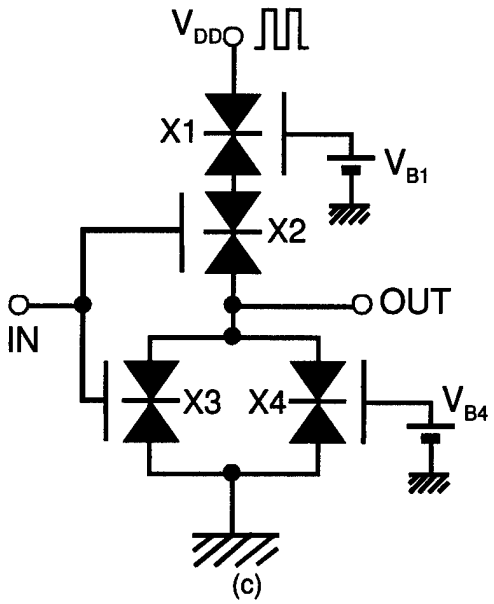


Figure 6. Literal circuit: (a) schematic cross-section, (b) microscope photograph, and (c) equivalent.

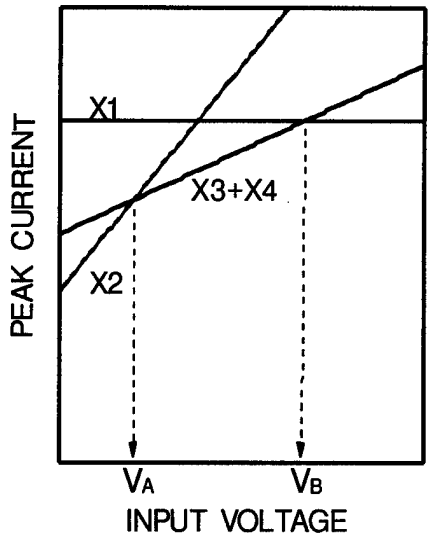


Figure 7. Relation of peak current for MJ-STTs in the proposed literal circuit.

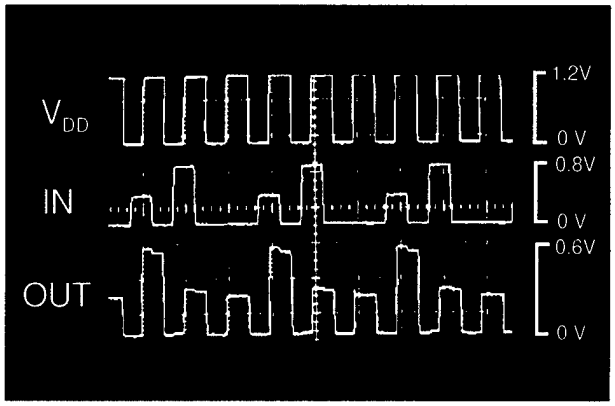


Figure 8. Input and output traces of the literal circuit.

The fabricated literal circuit displayed in Fig. 6 was designed to satisfy the relation shown in Fig. 7. To control the initial peak current and the increased peak current caused by the input voltage, the driver MJ-STT is split into two parts, that is, the parallel connection of X3 and X4. Figure 8 shows the input and output traces of the literal circuit depicted in Fig. 6. We chose a clocked supply voltage of 1.2 V. The input voltage was 0 V for the "0" logic level, 0.4 V for the "1" logic level, and 0.8 V for the "2" logic level. The literal function of x^{11}, which gives a high output level only for "1", is clearly demonstrated in Fig. 8. We also investigated this circuit by changing the bias voltage of V_{B1} and V_{B4}, and obtained the literal function of not only x^{11} but also x^{00} and x^{22}.

3. Multiple-valued memory using single electron transistor

Using oscillatory characteristics of single electron transistors (SETs), two types of multiple-valued memories were developed. Figure 9 shows one of the proposed memories (Si in-plane floating-dot memory). This device consists of a source, drain, SET island, floating dot, control gate, and back gate. Two artificial narrow constrictions between the SET island and source/drain act as tunneling barriers. The floating dot stores electrons injected from the control gate and acts as a memory node. This structure was fabricated using a 20-nm-thick highly-doped n-type Si film on a SIMOX (separation by implanted oxygen) wafer. To pattern a 10-nm scale fine-structure, we used calixarene, a high-resolution negative resist under electron-beam lithography, and reactive ion etching with CF_4 gas. The Si SET island was 30 nm in diameter and the floating dot was 15 nm in diameter. The Si surface is depleted and the narrow constriction between the island and source (or drain) nearly pinched off. The constriction thus acts as a tunneling barrier.

Electrical device characteristics were measured in a ^4He continuous-flow cryostat. The device was voltage-biased symmetrically between the source and drain at V_{SD} = 1 mV. A clear periodical current oscillation with back-gate and control-gate voltages (V_B and V_C) was observed (Fig. 10), and the formation of a high quality SET was indicated. These oscillations were observed up to 60 K. Within each oscillation period, increment/decrement of one electron in the SET island occurred by means of capacitive coupling with the gates. Based on these results, we can compensate the influence of gate voltages to the SET island by keeping the bias conditions of V_B = -0.54 V_C even during the gate sweep (shown in the upper curve of Fig. 10) [11]. However, when $|V_C|$ or $|V_{CB}|$ exceeded a certain threshold voltage, the current started to oscillate due to the electron injection from the control gate into the floating dot node by Fowler-Nordheim (FN) emission. These electrons change the

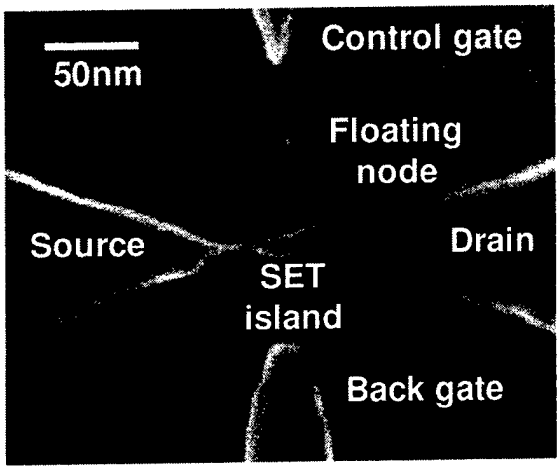


Figure 9. Scanning microscope photograph of a Si in-plane floating-dot memory.

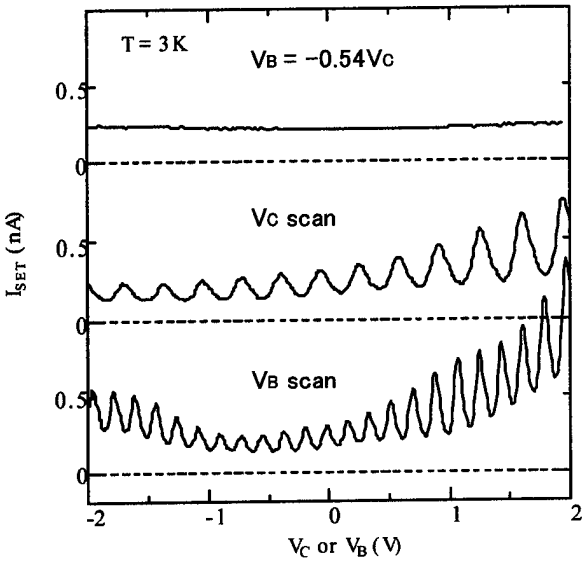


Figure 10. Current versus control-gate (back-gate) voltage: V_B scan while control gate is grounded, V_C scan while back gate is grounded, and control and back gates scan with V_B=-0.54V_C.

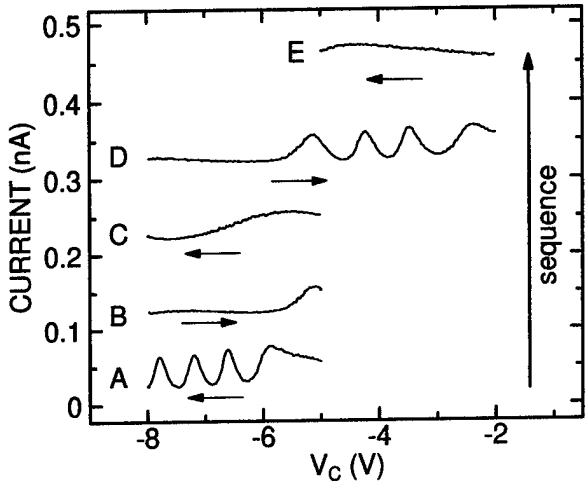


Figure 11. Current when V_C = -5 → -8 (A) → -5 (B) → -8 (C) → -2 (D) → -5 V (E) with V_B = -0.54 V_C. Each curve is shifted upward by 0.1 nA for clarity.

potential of the floating dot and also modify the charges on the SET island.

To demonstrate the memory function, we alternated the charged and discharged of the floating dot by sweeping the control and back gate voltages and kept the above condition (Fig. 11). Charging and discharging were seen as current oscillations at curves A and D. Once the floating dot was charged (curve A), a subsequent sweep of V_C in the same direction did not change the dot charge, and the current showed no oscillation (curve C). When $V_C > -5$ V, discharge occurred (curve D). This shows the memory function. If we could count the number of SET oscillations during charging and discharging of the floating dot by a count circuit, data storage could be multiple-valued. Since this type of reading is not disturbed by background charge, a reliable memory operation is also expected in the floating-dot memory devices with SET.

We have developed different types of multiple-valued memory device. Figure 12 shows a schematic view of the SiN SET memory device. The operation principle of this device is almost the same as the SET memory described above. Electrons are stored in the electron traps of the SiN films that are embedded in the gate insulator of the MOSFET instead of the floating node. Sensing the stored electrons are carried out by an Al based SET formed at the position of the gate electrode [11].

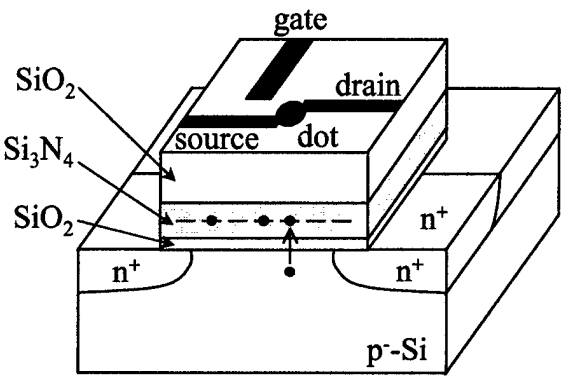


Figure 12. Schematic view of the SiN SET memory device.

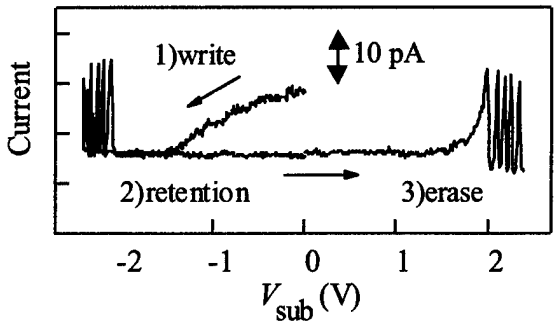


Figure 13. Write and erase operation of the SiN SET memory device.

Similar results were obtained with the planar SET memory, which indicates a proper operation of multiple-valued memory. In addition, nonvolatile characteristics were demonstrated as shown in Fig. 13. A retention time of about one hour was measured. This retention time is shorter than the expected value for the deep electron traps in the SiN films. By improving the fabrication process and the structural design, a longer retention time could be obtained.

4. Discussion

We described multiple-valued logic circuits using MJ-STTs and multiple-valued memory devices using SETs. These are the limited examples of the circuit application for multiple-valued logic using QFDs. Most of the QFDs have negative-differential-resistance characteristics and are considered very useful for implementing multiple-valued logic circuits. Multiple NDR characteristics are easily demonstrated by making the series connection of devices have single NDR characteristics. Using this approach, several multiple-valued circuits have been demonstrated using resonant-tunneling diodes. This approach is also applicable for other QFDs having NDR, interband tunneling diode, and resonant interband tunneling diode. MJ-STT is an example of a new device structure that is constructed with STT by this approach. Therefore, other QFDs also have a potential to implement the multiple-valued logic circuits.

To implement the multiple-valued logic circuits using QFDs, the development of design tools is very important. Since the current-voltage characteristics and materials are different from the conventional transistor, the design or simulation tools are not considered to be directly applicable to QFDs. A device simulator handling the quantum phenomena and circuit simulator, including the proper device model for QFDs, should be developed. The implementation of internal models of QFDs in the circuit simulator is also desired to speed up the circuit simulation.

The most important issue for practical usage of the QFDs is thought to be the development of new combination circuits or architecture for the QFDs utilizing the features of their current-voltage characteristics. However, since device researchers are not familiar with the system design and since system researchers are bit always

aware of new devices, discussions about new devices and new circuits between both groups are indispensable for paving the way for developing multiple-valued logic circuits using QFDs.

5. Conclusion

We demonstrated multiple negative-differential-resistance (NDR) characteristics in drain current-voltage characteristics using multiple-junction surface tunnel transistors (MJ-STTs). Multiple-valued logic gates of ternary inverter and literal circuits are implemented using the MJ-STTs. Oscillatory characteristics of drain current under gate modulation are showed by single electron transistors. Nonvolatile multiple-valued memory devices utilizing these characteristics to store electrons in floating nodes or electron traps are described and the fundamental operation of write and read of stored electrons are demonstrated.

Acknowledgments

This work was performed under the management of FED as a part of the MITI R&D Program (Quantum Functional Devices Project) supported by NEDO.

References

[1] H. Kawanami, "Introduction of Future Electron Devices Projects", Extended abstracts of 17th Symposium on Future Electron Devices, Tokyo, Japan, 1998, pp. 9-14.

[2] N. Yokoyama, K. Imamura, S. Muto, S. Hiyamizu and H. Nishi, "A new functional, resonant-tunneling hot electron transistor (RHET)," Jpn. J. Appl. Phys., vol. 24, no. 11, pp. L853-L854, 1985.

[3] K. Imamura, M. Takatsu, T. Mori, T. Adachihara, H. Ohnishi, S. Muto, and N. Yokoyama, "Arithmetic logic circuits using resonant-tunneling hot electron transistors (RHETs)," Extended abstract of 1991 Int. Conf. Solid State Devices and Materials, pp. 80-82.

[4] S. Sen, F. Capasso, A. Y. Cho, and D. Sivco, "Resonant tunneling device with multiple negative differential resistance: Digital and signal applications with reduced circuit complexity," IEEE Trans. Electron Devices, vol. ED-34, pp. 2185-2191, 1987.

[5] T. Waho, K. J. Chen, and M. Yamamoto, "A novel multiple-valued logic gate using resonant tunneling devices," IEEE Electron Device Lett., vol. EDL-17, no. 5, pp. 223-225, 1996.

[6] T. Baba, "Proposal for Surface Tunnel Transistors," Jpn. J. Appl. Phys., vol. 31, no. 4B, pp. L455-L457, 1992.

[7] T. Baba and T. Uemura, "Multiple-junction surface tunnel transistors for multiple-valued logic circuits," Proceedings of 27th Int. Symp. Multiple-Valued Logic, Nova Scotia, Canada, 1997, pp. 41-46.

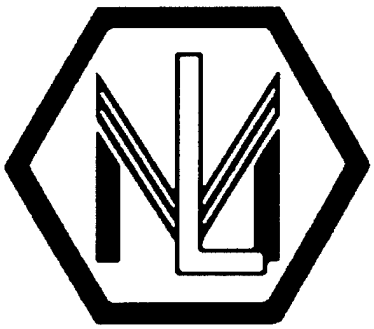
[8] T. Baba and T. Uemura, "Development of InGaAs-based multiple-junction surface tunnel transistors for multiple-valued logic circuits," Proceedings of 28th Int. Symp. Multiple-Valued Logic, Fukuoka, Japan, 1998, pp. 7-12.

[9] H. Grabert and M. H. Devoret, "Single charge tunneling - Coulomb blockade phenomena in nanostructures", NATO ASI Series, Series B: Physics vol. 294, Plenum Press, New York, 1992.

[10] K. Maezawa and T. Mizutani, "New resonant tunneling logic gate employing monostable-bistable transition," Jpn. J. Appl. Phys., vol. 32, no. 1A-B, pp. L42-44, 1993.

[11] C. D. Chen, Y. Nakamura, and J. S. Tsai, "Aluminum single-electron nonvolatile floating gate memory cell", Appl. Phys. Lett. vol. 71, pp. 2038-2040, 1997.

SESSION IIA
ALGEBRA I
CHAIR: Ivo Rosenberg



MULTIVALUED BINARY RELATIONS and POST ALGEBRAS.

by Michel SERFATI.
L.I.A.F.A, Université Paris VII (Denis Diderot)
and CNRS (ERS 586). Case 7014,
2 Place Jussieu,75251 Paris Cedex 05.
E-mail: serfati@cicrp.jussieu.fr

ABSTRACT : This paper deals with a multivalued extension of the concept of a binary relation on a set E. If R is such an r-valued relation on E, for every (x, y) ∈ E^2, there exists exactly one i ∈ {0, 1, ..., r-1} such that the degree of comparability *of (x, y) with respect to R is equal to i. The set of all r-valued relations on E is then equipped with an order relation ≤ and turns out to be a Post algebra of order r. This first part of the paper makes a large use of the results of a previous paper (1999 a). In a second step, we study multivalued product of relations, as well as multivalued reflexivity, symmetry, antisymmetry, and transitivity. It follows for example that, from a multivalued viewpoint, symmetry and antisymmetry turn out to be quite different, since the latter is hereditary while the former is not. We finally investigate some properties of multivalued equivalences, multivalued orders and preorders.*

1. Preliminaries.

§ 1. 1 *Center of a bounded distributive lattice.*

We will use the following notations : if one has x ≤ y in a poset, we will say that y *dominates* x. Supremum and infimum of two elements x and y of a lattice T will be denoted by x ∨ y and x ∧ y. If it exists, the supremum (resp. infimum) of any indexed family x_i ($i \in I$) of elements of T will be denoted by $\bigvee_{i \in I} x_i$ (resp. $\bigwedge_{i \in I} x_i$) . Universal elements (if they exist) will be denoted by 0 and 1 ; we shall also say that T is a (0,1) - lattice. Recall that in any (0,1) - distributive lattice, the set of the complemented elements is a Boolean algebra, called the *center* of T. The complement of any element a of a boolean algebra will be denoted by $\bar{a}$.

§ 1. 2 *Binary relations.*

Recall that a binary relation R on a set E is simply a subset of E^2, so that the set of all the binary relations on E is the powerset $\mathcal{P}(E^2)$, which is usually ordered by inclusion : $R \subset S$ if $(x, y) \in R \Rightarrow (x, y) \in S$. Moreover $(\mathcal{P}(E^2) ; \cup , \cap , \bar{}, \varnothing, E^2)$ is a boolean algebra, so that we can define join, meet, and complementation for binary relations : $R \cup S$; $R \cap S$; $\bar{R}$. The relation $\bar{R}$ is called the *complement* of the relation R : $(x, y) \in \bar{R}$ if and only if $(x, y) \notin R$. Recall also the *diagonal* Δ of E^2 is the binary relation : $\Delta = \{ (x, x) ; x \in E \}$. Since all relations considered in this paper will be binary, from now we shall omit this term and simply speak of "relations". Recall now that every powerset $\mathcal{P}(\Omega)$ is isomorphic to $<2>^{\Omega}$ *via* the boolean characteristic function theorem ($<2> = \{0,1\}$). We will apply first this result to $\Omega = E^2$, so that the set $\mathcal{P}(E^2)$ of all relations on E is boolean-isomorphic to $< 2 >^{(E^2)}$. In fact to every relation R on E, we associate the function $\mathcal{K}(R)$ from E^2 to <2> (also denoted $\underline{R}$) by :

$(\forall (x, y) \in E^2))$ $\underline{R}(x, y) = 1$ if and only if $(x, y) \in R$ (consequently : $\underline{R}(x, y) = 0$ if and only if $(x, y) \notin R$). This theorem expresses this very simple fact : if a R is a given relation on E, and (x, y) any element in E^2, only two cases may happen : either (x, y) belongs to R, either it does not. Consequently, from a logical viewpoint, the concept of "relation " is here a two-valued one. The aim of this paper is to extend this concept to the multivalued case.

§ 1. 3 *Subdiagonals.*

Let E be any set and k∈IN*(i.e any non-zero natural integer). Define the *subdiagonal* of order k of $\mathcal{P}(E^2)$ as the set of all sequences of k relations on E, subject to the decreasing chain condition, that is :

$\mathbf{k}(E^2) = \{ R = (R_{<1>}, ..., R_{<k>}) \in [\mathcal{P}(E^2)]^k / R_{<j>} \supset R_{<j+1>} ; 1 \le j \le k-1 \}$

It is easily proved ([6], Theorem 1) that, equipped with the product order (componentwise), $\mathbf{k}(E^2)$ is a complete bounded distributive lattice, and also that an element R of $\mathbf{k}(E^2)$ is complemented if and only if all its components $R_{<k>}$ are equal, so that the center $\Delta\mathbf{k}(E^2)$ of $\mathbf{k}(E^2)$ is boolean isomorphic to the set $\mathcal{P}(E^2)$ of all relations on E, under the map $\mathcal{P}(E^2) \to \Delta\mathbf{k}(E^2) : R \to$ "R" = (R, R, ..., R).

§ 1. 4 *Post algebras.*

We recall now the Epstein-Traczyk's definition (see [2] and [8]) of a Post algebra : given any integer $r \ge 2$, a Post algebra of order r is a bounded distributive lattice P, whose center is denoted by B, such that :

1. There exists in P a finite chain $< r >$ with r elements : $< r > = \{ 0 = e_0 < e_1 < ..... < e_{r-1} = 1 \}$.

2. To every element x of P, one can associate a family $(x^{<i>})_{0 \le i \le r-1}$ of r elements of B subject to "orthonormality" conditions (1.4.1):

0-7695-0161-3/99 $10.00 © 1999 IEEE

$\bigvee_{i=0}^{r-1} x^{<i>} = 1$ and $x^{<i>} \wedge x^{<j>} = 0$ (for all $i \neq j$)

such that : $x = \bigvee_{i=0}^{r-1} x^{<i>} \wedge e_i$ (1.4.2)

3. The representation (1.4.2) is unique in the following sense : if x happens to admit the representation $x = \bigvee_{i=0}^{r-1} a^i \wedge e_i$ where $a^0, a^1, .., a^{r-1} \in B$, are subject to orthonormality conditions (1.4.1), then $x^{<i>} = a^i$ for every i ◆

It is easily shown (e.g [1]) that, if P is a Post algebra of order r, then the integer r is unique, as well as the chain $< r >$ and, for every x, the disjunctive components $x^{<i>}$'s. One can also prove that the subdiagonal $(r\text{-}1)(E^2)$ (cf §1.3 above) is a simple example of complete Post algebra of order r (see[6]).

2. Multivalued relations.Quasi-relations.

In sections 2-5 below we shall use of some of the results of [6], where we proved many theorems concerning $\mathfrak{P}_r(\Omega)$, that is the lattice of all r-ordered partitions of some set Ω. We will recall below without proofs some of these results, and apply them to the case $\Omega = E^2$, that is, we shall translate them in a "relational" context.

Definition 2.1 ([6], Definition 2.1) Let E be any set and $r \in \mathbb{N}^*$, $r \geq 2$. We call a *multivalued relation of valuation* r *on E*, (briefly mvr), every family $R = (R^{<0>}, ..., R^{<r-1>})$ of r relations on E such that :

$$\bigcup_{0 \leq i \leq r-1} R^{<i>} = E^2 \quad \text{and} \quad R^{<i>} \cap R^{<j>} = \emptyset$$

for all $i \neq j$.

In other words, a mvr of valuation r is simply a partition of the set E^2 by the means of r of its subsets, what we called in [6] an r-*ordered partition* of E^2. Therefore, for every ordered pair (x, y) in E^2, there exists exactly one i = 0, 1, .., r-1 such that $(x, y) \in R^{<i>}$; i (= $i_R(x, y)$) is the *degree of comparability* of x to y with respect to R. The $R^{<i>}$ will be called the *disjunctive components* of R (there are r such components).

If R is a mvr on E of valuation r, we will also say that R is an r-*valued relation* on E. For the sake of a proper distinction between the multivalued case and the previous one, it will be convenient from now on to call "common relations" on E what we had simply called "relations" that is, elements of $\mathfrak{P}(E^2)$. Note that, whatever relations may be, either multivalued or common, they will remain "binary" anyway, so that we will go on omitting this term. An important special case happens when all intermediary relations ($R^{<i>}$; $1 \leq i \leq r - 2$) are empty. It follows that such a mvr must have the value : $(\bar{\rho}, \emptyset, ..., \emptyset, \rho) = ' \rho '$ for some (common) relation ρ on E. (' ρ ' to be read as «quasi-ρ»). ' ρ ' is called a *quasi-common multivalued relation* on E (of valuation r), and, for short, a *quasi-relation on* E. It is clear that, if R =' ρ ', then for every (x, y) in E^2 the degree of comparability of x to y belongs to the pair {0, r-1} ; a multivalued relation which is not a quasi-relation is said *actual*. A multivalued relation R such that $R^{<0>} = \emptyset = R^{<r-1>}$ is said *strictly actual* (its non empty components are all intermediary ones) ◆

Note that, for r = 2, $\mathfrak{P}_2(E^2) = \{(\bar{R}, R) ; R \in \mathfrak{P}(E^2)\}$. It follows that the power set $\mathfrak{P}(E^2)$ of all common relations on E is isomorphic -under the map : $R \rightarrow (\bar{R}, R)$- to the lattice $\mathfrak{P}_2(E^2)$ of all mvrs on E of valuation 2.

3. Post algebra structure on $\mathfrak{P}_r(E^2)$.

§ 3. 1 *Lattice structure on the set of multivalued relations.*

Let $R = (R^{<i>})\ 0 \leq i \leq r - 1$ and $S = (S^{<i>})\ 0 \leq i \leq r-1$ be any two mvrs on E. Define $R \leq S$ by :

$$(\forall\ i \in \{0, 1, ..., r-1\})\ R^{<i>} \subset \bigcup_{j=i}^{r-1} S^{<j>}$$

then, one can prove ([6], Theorem 2.1) that ≤ is an order relation on $\mathfrak{P}_r(E^2)$. Hence, according to this definition, a multivalued relation R precedes S if and only if, for every $(x, y) \in E^2$, the degree of comparability of (x, y) *w.r.t* S is at least the the degree of comparability of (x, y) *w.r.t* R. We then proved ([6], Theorem 2.3) that $\mathfrak{P}_r(E^2)$, equipped with this order relation is a complete bounded distributive lattice: for every pair R = $(R^{<i>})\ 0 \leq i \leq r - 1$ and $S = (S^{<i>})\ 0 \leq i \leq r - 1$ of elements of $P_r(E^2)$, one has:

$$[R \vee S]^{<i>} = \left[R^{<i>} \cap \bigcup_{j=0}^{i} S^{<j>}\right] \cup \left[S^{<i>} \cap \bigcup_{j=0}^{i} R^{<j>}\right]$$

$$[R \wedge S]^{<i>} = \left[R^{<i>} \cap \bigcup_{j=i}^{r-1} S^{<j>}\right] \cup \left[S^{<i>} \cap \bigcup_{j=i}^{r-1} R^{<j>}\right]$$

for all i = 0,1,..., r-1.

In other words, an element (x, y) of E^2 has its degree of comparability *w.r.t* R ∨ S equal to i if and only if alternatively, either it has degree i *w.r.t* R and degree at most equal to i *w.r.t* S, either it has degree i *w.r.t* R and degree at most equal to i *w.r.t* S. We also proved that a multivalued relation is complemented in the lattice $\mathfrak{P}_r(E^2)$ if and only if it is a quasi-relation. The center $\Delta_r(E^2)$ of $\mathfrak{P}_r(E^2)$ is boolean-isomorphic to $\mathfrak{P}(E^2)$ *via* the isomorphism : $\rho \rightarrow {}'\rho'$. So, we will be able to identify, in the lattice-theoretic sense, a common relation ρ and the quasi-relation 'ρ ' ; when restricted to quasi-relations, *sup* and *inf* so coincide, up to isomorphism, with set-theoretic meet and join for (common) relations.

Lemma : From the formulas above, we get :

$(R \vee S)^{<0>} = R^{<0>} \cap S^{<0>}$

$(R \wedge S)^{<0>} = R^{<0>} \cup S^{<0>}$

$(R \vee S)^{<r-1>} = R^{<r-1>} \cup S^{<r-1>}$

$(R \wedge S)^{<r-1>} = R^{<r-1>} \cap S^{<r-1>}$

§ 3. 2 *Fundamental chain ; the postian structure for the lattice of multivalued relations.*

In a further step ([6], th.2.7), we introduced in $\mathfrak{P}_r(E^2)$, what we called a fundamental chain in the following way : $(e_0, e_1, ..., e_{r-1})$ is the r-tuple of elements of $[\mathfrak{P}(E^2)]^k$ defined for $0 \le i \le r-1$, by : $(e_i)_j = E^2$ if $i = j$ and $(e_i)_j = \emptyset$ otherwise, that is : $e_i = (\emptyset, \emptyset, ... \emptyset, E^2, \emptyset, ..., \emptyset)$. Clearly, every e_i is a mvr (for every $(x, y) \in E^2$, the degree of comparability of x to y is equal to i) ; moreover, with respect to the order of $\mathfrak{P}_r(E^2)$, the e_i 's form a chain with r elements : $0 = e_0 < e_1 < ... < e_{r-1} = 1$ ◆

On the other hand, recall that every $R_{<k>}$ is a common relation, so that ' $R_{<k>}$ ' is a quasi -relation. Then one can prove ([6], th. 2.9) the first fundamental representation theorem:

Theorem 1 (disjunctive representations) : **For every multivalued relation R of valuation r** : $R = (R^{<i>})\ 0 \le i \le r-1 \in \mathfrak{P}_r(E^2)$, **one has :**

$$R = \bigvee_{0 \le i \le r-1} {}'R^{<i>}{}' \wedge e_i$$

Moreover, such a representation is unique in the following sense :

If one has $R = \bigvee_{0 \le i \le r-1} {}'X^i{}' \wedge e_i$

where the $'X_i'$ **s are r quasi-relations such that : such that** $\bigvee_{0 \le i \le r-1} {}'X^i{}' = 1 \quad (= (\emptyset, ..., \emptyset, E^2))$ **and** $'X^i' \wedge 'X^j' = 0$ $(= (E^2, \emptyset, ..., \emptyset))$ **for all** $i \ne j$, **then** $X^i = R^{<i>}$ **for every** $i = 0, 1, ..., r-1$ ◆

From Theorem 1, it follows this important result ([6], corollary 2.8.1) :

Theorem 2 : **For every set** E **and every** $r \ge 2$, $\mathfrak{P}_r(E^2)$ **is a complete Post algebra of order r.**

It follows that the present theory is actually a set-theoretic realization of multivalued logic Post conceptions. Since $\mathfrak{P}_r(E^2)$ is complete, it follows also that, for every family $(S_i)\ i \in I$ of mvrs on E, then $\bigvee_{i \in I} S_i$ and $\bigwedge_{i \in I} S_i$ exist.

§ 3. 3 *Monotonic components of a multivalued relation. First isomorphism theorem.*

In a further step, one define ([6], Theorem 2.2, formulas 2.7), for every mvr $R = (R^{<i>})\ 0 \le i \le r-1 \in \mathfrak{P}_r(E^2)$, the sequence of its (r-1) *monotonic components* $R_{<i>} = \bigcup_{j=i}^{r-1} R^{<j>}$ for $i = 1, 2, ..., r-1$.

Actually, to R we associate $\mathfrak{D}(R) = (R_{<1>}, R_{<2>}, ..., R_{<r-1>}) \in [\mathfrak{P}(E^2)]^{r-1}$ by : Then one can prove ([6], Theorem 2.2) :

Theorem 3 (First fundamental isomorphism theorem) : $\mathfrak{D}$ **is a lattice isomorphism from** $(\mathfrak{P}_r(E^2), \le)$ **onto** $((r-1)(E^2), \le)$, **the inverse mapping** $\mathfrak{D}^{-1}$ **being defined by** :

$R^{<0>} = \overline{R_{<1>}}$; $R^{<j>} = R_{<j>} \cap \overline{R_{<j+1>}}$ for $1 \le j \le r-2$

$R^{<r-1>} = R_{<r-1>}$ ◆

Remarks 3.3.1 Note that $(r-1)(E^2)$ is the subdiagonal of order (r-1) of E^2, as defined *supra* in § 1.3. Note also that the "one-to-one" part of the proof is usual in measure and probability theory : to every family of r pairwise disjoint relations on E, we associate, by a one-to-one correspondance, a decreasing sequence of (r-1) relations. For instance, for r = 3 and $R = (R^{<0>}, R^{<1>}, R^{<2>})$, one has :

$\mathfrak{D}(R) = (R^{<1>} \cup R^{<2>}, R^{<2>}) = (R_{<1>}, R_{<2>})$

and conversely, with $R_{<1>} \supset R_{<2>}$, we get :

$\mathfrak{D}^{-1}(R_{<1>}, R_{<2>}) = (\overline{R_{<1>}}, R_{<1>} \cap \overline{R_{<2>}}, R_{<2>}) = (R^{<0>}, R^{<1>}, R^{<2>})$.

Remarks 3.3.2 From Theorem 3, it also follows that $R \le S$ (resp. $R = S$) if and only if $R_{<k>} \le S_{<k>}$ (resp. $R_{<k>} = S_{<k>}$) for $k = 1, 2, ...,$ r-1. It also follows that $[R \vee S]_{<k>} = R_{<k>} \vee S_{<k>}$ and $[R \wedge S]_{<k>} = R_{<k>} \wedge S_{<k>}$. On the other hand, let $\mathfrak{D}_1$ be the restriction of $\mathfrak{D}$ to the respective centers of the two lattices, $\mathfrak{P}_r(E^2)$and $(r-1)(E^2)$; then $\mathfrak{D}_1$ is a boolean isomorphism of these centers. We then get : $\mathfrak{D}('\rho') = "\rho"$ (for the definition of "ρ", see above §1.3), for every common relation ρ on E. It follows that a mvr R is a quasi-relation if and only if all its monotonic components $R_{<k>}$ are equal.

Theorem 4 (Monotonic representations) : **For every multivalued relation** $R \in \mathfrak{P}_r(E^2)$, **one has** :

$$R = \bigvee_{1 \le i \le r-1} {}'R_{<i>}{}' \wedge e_i$$

Moreover, such a representation is unique in the following sense :

If one has $R = \bigvee_{1 \le i \le r-1} {}'X_i{}' \wedge e_i$

where the $'X^i'$ **s are** (r-1) **quasi-relations such that** $'X_i' \ge 'X_{i+1}'$ **for every i, then** $X_i = R_{<i>}$ **for every** $i = 1, 2, ...,$ r-1.

4. Negation and pseudocomplement of a multivalued relation.

§ 4.1 *Negation.*

We also proved in ([6], Theorem 2.4) that we could equip $\mathfrak{P}_r(E^2)$ with an involutive lattice duality by associating to every mvr R of valuation r : $R = (R^{<i>})\ 0 \le i \le r-1 \in \mathfrak{P}_r(E^2)$, the r-tuple $R^{\otimes}$ defined by : $(R^{\otimes})^{<i>} = R^{<r-1-i>}$ for $0 \le i \le r-1$ (one has inverted the order of the components, hence the weight of the comparabilities) : $R^{\otimes}$ will be called the *negation* of R. We have, for every R and S : $(R \vee S)^{\otimes} = R^{\otimes} \wedge S^{\otimes}$; $(R \wedge S)^{\otimes} = R^{\otimes} \vee S^{\otimes}$; $(R^{\otimes})^{\otimes} = R$. If R is a quasi-relation, say R = ' ρ ', so is $R^{\otimes}$ = ' $\bar{\rho}$ '.

4.2 *Pseudocomplements.*

We also proved in ([6], Theorem 2.5) that $\mathfrak{P}_r(E^2)$ is a Heyting algebra, that is, to say that, for every (R, S) $\in [\mathfrak{P}_r(E^2)]^2$, the inequation : $R \wedge X \le S$ admits a greatest solution in $\mathfrak{P}_r(E^2)$, denoted (S | R), and called *inf-relative pseudo-complement* of S to R. In fact, formulas :

$$(S \mid R)_{<j>} = \bigcap_{1 \le i \le j} (S_{<i>} \cup \overline{R_{<i>}}) \text{ for every } j$$

provide the monotonic components of (S | R) (cf.[5]). Dually, we proved existence and unicity for the *sup-relative pseudo-complement*, that is, the least solution of $R \vee X \ge S$. We may then conclude, as in ([6], Theorem. 2.6) that, for every R in $\mathfrak{P}_r(E^2)$, the equation $R \wedge X = 0$ admits a greatest solution R*, which is simply the quasi-relation : $R^* = ' R^{<0>} '$. The map $R \to R^*$ is a lattice duality such that $R = 0 \Leftrightarrow R^* = 1$. Hence $(R \vee S)^* = R^* \wedge S^*$; $(R \wedge S)^* = R^* \vee S^*$ ◆

Finally note that, if R is a quasi-relation, say R = ' ρ ', so is R* = ' $\bar{\rho}$ '. Moreover, for every mvr R, one has : $(R^{\otimes})^* = (R^*)^{\otimes} = ' \overline{R^{<0>}} '$

5. Characteristic functions. Second isomorphism theorem.

Let < r > be any chain with r elements : < r > = $\{a_0 = 0 < a_1 < .. < a_{r-1} = 1\}$, *e.g* : $a_k = \frac{k}{r-1}$ $(0 \le k \le r-1)$. Then, for every mvr $R = (R^{<i>})\ 0 \le i \le r-1 \in \mathfrak{P}_r(E^2)$, we may associate $\mathcal{K}(R) \in <r>^{(E^2)}$ by : $(\forall (x, y) \in E^2)\ \mathcal{K}(R)(x, y) = a_j \Leftrightarrow (x, y) \in R^{<j>}$.

We proved easily ([6], Th. 2.10) that $\mathcal{K}(R)$ is well defined on E^2 ; $\mathcal{K}(R)$ is the *(multivalued) characteristic function* of R and will be denoted from now $\underline{R}$. We then proved :

Theorem 5 (Second fundamental isomorphism theorem) **For every set** E **and every integer** $r \ge 2$, $\mathfrak{P}_r(E^2)$ **is lattice-isomorphic to** $<r>^{(E^2)}$, ***via*** **the mapping** $\mathcal{K}$.

Remarks 5.1 It follows that : Card $(\mathfrak{P}_r(E^2))$ = $<r>^{((\text{Card}\,E)^2)}$ = Card $[(r-1)(E^2)]$. In the finite case, with Card(E) = n, then Card $(\mathfrak{P}_r(E^2)) = r^{(n^2)}$. It follows also that, for every family $(S_i)\ i \in I$ of mvrs on E, one has : $\underline{\left(\bigvee_{i \in I} S_i\right)} = \max_{i \in I} \underline{S_i}$ and

$\underline{\left(\bigwedge_{i \in I} S_i\right)} = \min_{i \in I} \underline{S_i}$

6. Product of multivalued relations.

Recall first that if P and Q are two common relations on a set E, one usually defines the *product relation* $P \otimes Q$ in the following way : for every $(x, y) \in E^2$: $(x, y) \in P \otimes Q$ if and only if there exists $z \in E$ such that $(x, z) \in P$ and $(z, y) \in Q$. Hence $P \otimes Q$ is also a common relation. The product of common relations is shown to be associative and to have some other properties (cf. for instance [3])

Definition 6.1 : Let R and S be any two multivalued relations on E. Define $R \otimes S$ as the multivalued relation whose characteristic function $\underline{R \otimes S}$ has the value :

$$\underline{R \otimes S}(x, y) = \bigvee_{z \in E} \underline{R}(x, z) \wedge \underline{S}(z, y) \quad (6.1)$$

for every $(x, y) \in E^2$ ◆

To understand this definition, recall (see Theorem 5) that the map $R \to \underline{R}$ is a one-to-one correspondence and also a lattice isomorphism, so that whenever R and S are given, then $\underline{R}$ and $\underline{S}$, which are also given, are both maps from E^2 onto <r>. Since <r> is a complete lattice, it follows that the right part of relation (6.1) defines also some map G from E^2 onto <r>. By a repeated use of Theorem 5, there exists exactly one mvr T such that G = $\underline{T}$. We denote it by $T = R \otimes S$, and called it the *product of the multivalued relations* R and S.

Theorem 6 : **Let** R **and** S **be any two quasi relations on** E, **say** R = ' ρ ' **and** S = ' μ '. **Then** $R \otimes S$ ***is a quasi relation and***

$$' \rho ' \otimes ' \mu ' = ' \rho \otimes \mu '$$

Proof : Recall that $a_0 = 0$ and $a_{r-1} = 1$. Since R = ' ρ ' = $(\bar{\rho}, \emptyset, ..., \emptyset, \rho)$ and S = ' μ ' = $(\bar{\mu}, \emptyset, ..., \emptyset, \mu)$, both ρ(x,y) and μ(x, y) belong to {0,1}. From relation (6.1), it then follows that $\underline{R \otimes S}(x, y) \in \{0,1\}$ for every (x, y) in E^2. Hence the mvr $\underline{R \otimes S}$ is a quasi-

relation. Moreover, from relation (6.1), it follows that $\underline{R \otimes S}(x, y) = 1 = (\text{`}\rho\text{'} \otimes \text{`}\mu\text{'})(x, y)$ if and only if there exists z in E such that $\underline{R}(x, z) = 1$ and $\underline{S}(z, y) = 1$, that is $(x, z) \in \rho$ and $(z, y) \in \mu$. In other words $(\text{`}\rho\text{'} \otimes \text{`}\mu\text{'})(x, y) = 1$ if and only if (x, y) belongs to the (common) relation $\rho \otimes \mu$. Hence $(\text{`}\rho\text{'} \otimes \text{`}\mu\text{'})^{<r-1>} = \rho \otimes \mu$. Since $(\text{`}\rho\text{'} \otimes \text{`}\mu\text{'})(x, y) \in \{0,1\}$, it follows that $(\text{`}\rho\text{'} \otimes \text{`}\mu\text{'})^{<j>} = \emptyset$ for $1 \le j \le r-2$, and also $(\text{`}\rho\text{'} \otimes \text{`}\mu\text{'})(x, y) = 0$ if and only if $(x, y) \in \overline{\rho \otimes \mu} = (\text{`}\rho\text{'} \otimes \text{`}\mu\text{'})^{<0>}$. Finally $\text{`}\rho\text{'} \otimes \text{`}\mu\text{'} = \text{`}\rho \otimes \mu\text{'}$ ◆ Therefore the product of mvrs turns out to be a proper extension of the product of common relations as recalled *supra*.

Remark 6.1 : Assume E to be a finite set with n elements, say $E = \{1, 2,.., n\}$ and R to be a mvr on E. Define the square postian matrix A of size n, whose element A_{ij} in i-th row and j-th column is equal to R (i, j) $\in <r>$. With our notations in ([5]), we write $A \in M_n(<r>)$. A is said to be the *matrix of the multivalued relation* R (denoted A = Mat (R)). It is easily shown that if A = Mat (R), and B= Mat (S), then A o B = Mat (R $\otimes$ S), where o denotes the postian matrix product (see [5]) ◆ We will now study some properties of the product of mvrs, and will leave some of the proofs to the reader.

Theorem 7 (Associativity) : **For all** R, S, T $\in \mathcal{P}_r(E^2)$ $(R \otimes S) \otimes T = R \otimes (S \otimes T)$ (6.2)
(the proof is left to the reader)

Theorem 8 (Order compatibility). **Let** R, S, T **be any three multivalued relations and** R≤ S.**Then** $R \otimes T \le S \otimes T$ (6.3) **and** $T \otimes R \le T \otimes S$ (6.4)

Proof : $R \le S$ implies $\underline{R} \le \underline{S}$, so that $\underline{R}(x, y) \le \underline{S}(x, y)$ for every (x, y) in E^2. It follows : $\underline{R \otimes T}(x, y) = \bigvee_{z \in E} \underline{R}(x, z) \wedge \underline{T}(z, y)$

$\le \bigvee_{z \in E} \underline{S}(x, z) \wedge \underline{T}(z, y) = \underline{S \otimes T}(x, y)$, so that (6.3) holds. An analagous proof is valid for (6.4).

Theorem 9 (Distributivities). **Let** $(S_i)\ i \in I$ **be any family of multivalued relations on E. Then for any multivalued relation** R, **one has:**

$$R \otimes \left(\bigvee_{i \in I} S_i \right) = \bigvee_{i \in I} (R \otimes S_i) \qquad (6.5)$$

$$\left(\bigvee_{i \in I} S_i \right) \otimes R = \bigvee_{i \in I} (S_i \otimes R) \qquad (6.6)$$

$$R \otimes \left(\bigwedge_{i \in I} S_i \right) \le \bigwedge_{i \in I} (R \otimes S_i) \qquad (6.7)$$

$$\left(\bigwedge_{i \in I} S_i \right) \otimes R \le \bigwedge_{i \in I} (S_i \otimes R) \qquad (6.8)$$

(the proof is left to the reader : recall that $\mathcal{P}_r(E^2)$ is complete, so that $\bigwedge_{i \in I} S_i$ exists for every $(S_i)\ i \in I$).

Theorem 10 (Unit element). Let I denote the quasi-relation I = ' Δ '. **Then, for every multivalued relation** R :

$$I \otimes R = R \otimes I = R \qquad (6.9)$$

Proof : For every (x, y) in E^2 : $\underline{I \otimes R}(x, y) = \bigvee_{z \in E} \underline{I}(x, z) \wedge \underline{R}(z, y)$
On the other hand $\underline{I}(x, z) \in \{0, 1\}$ and $\underline{I}(x, z) = 1$ iff $(x, z) \in \Delta$, that is, x = z.
Hence, for every (x, y) in E^2 : $\underline{I \otimes R}(x, y) = \underline{R}(x, y)$ and $I \otimes R = R$.
Hence I is the unit element of $(\mathcal{P}_r(E^2), \otimes)$.

Definition 6.2 : A mvr R is said to be *reflexive* if $I \le R$.
Remarks 6.2 : Since I = ' Δ ', R is reflexive iff $(x, x) \in R^{<r-1>}$ for every $x \in E$. Consequently, if R is a quasi-relation, say R = ' ρ ', then R is reflexive iff $(x, x) \in \rho$ for every $x \in E$. Hence ' ρ ' is a reflexive mvr iff ρ is a common reflexive relation.

7. Transitivity for multivalued relations.

§ 7. 1 *Transitivity.*

We first state a lemma:
Lemma 7.1 : Let f be the characteristic function of a multivalued relation R (see Theorem 5).Then, for every k= 1, 2,..., r-1, the monotonic component $f_{<k>}$ is the characteristic function of $'R_{<k>}'$.

Proof : Recall first that $f(x, y) \in <r>$, so that $f_{<k>}(x, y) = (f(x, y))_{<k>} \in <2>$. In fact,
-. if $f(x, y) = a_0 = 0$, then $f_{<k>}(x, y) = 0$ for every k.
-. if $f(x, y) = a_j$, with $1 \le j \le r-2$, then $f_{<k>}(x, y) = 1$ for $1 \le k \le j$, and $f_{<k>}(x, y) = 0$ for $j + 1 \le k \le r-1$.
-. if $f(x, y) = a_{r-1} = 1$, then $f_{<k>}(x, y) = 1$ for every k.
Hence $f_{<k>}(x, y) = 1 \Leftrightarrow f(x, y) = a_j$ for $j \ge k \Leftrightarrow (x, y) \in R^{<j>}$ for $j \ge k \Leftrightarrow (x, y) \in \bigcup_{k \le j \le r-1} R^{<j>} = R_{<k>}$.
Since $f_{<k>}(x, y) \in \{0,1\}$, it follows that $f_{<k>}(x, y) = 0 \Leftrightarrow (x, y) \in \overline{R_{<k>}}$. It follows that $f_{<k>}$ is the characteristic function of the mvr $(\overline{R_{<k>}}, \emptyset, ..., \emptyset, R_{<k>})$, that is $'R_{<k>}'$ ◆ Finally, with our notations, we get : if $f = \underline{R}$, then $f_{<k>} = \overline{'(R)_{<k>}'}$ ◆

Define now $R^1 = R$ and, by induction : $R^{n+1} = R^n \otimes R$.

Definition 7.1 : A multivalued relation R is said to be *transitive* if $R^2 \leq R$

Theorem 11 : Let R be any given multivalued relation. Then R is transitive if and only if each of its monotonic components $R_{<k>}$ is (commonly) transitive.

Proof : R is transitive if and only if $R^2 \leq R$, that is,for every (x, y) in E^2 :

$$\bigvee_{z \in E} \underline{R}(x,z) \wedge \underline{R}(z,y) \leq \underline{R}(x,y) \quad (7.1)$$

From remarks (3.3.2), it follows that (7.1) is equivalent to (7.2) below:

$$\left(\bigvee_{z \in E} \underline{R}(x,z) \wedge \underline{R}(z,y) \right)_{<k>} \leq \left(\underline{R}(x,y) \right)_{<k>}$$

for every k = 1, 2,.., r-1. On the one hand, by the definition of $\underline{R}$: $(\underline{R}(x, y))_{<k>} = ((R)_{<k>})(x, y)$.

On the other hand, since every $A \rightarrow A_{<k>}$ is a lattice homomorphism (see remarks 3.3.2) :

$$\left(\bigvee_{z \in E} \underline{R}(x,z) \wedge \underline{R}(z,y) \right)_{<k>}$$
$$= \bigvee_{z \in E} \left(\underline{R}(x,z) \right)_{<k>} \wedge \left(\underline{R}(z,y) \right)_{<k>}$$
$$= \bigvee_{z \in E} \left(\left(\underline{R} \right)_{<k>}(x,z) \right) \wedge \left(\left(\underline{R} \right)_{<k>}(z,y) \right)$$

Thus the relation (7.2) turns out to be equivalent to : $\bigvee_{z \in E} \left(\left(\underline{R} \right)_{<k>}(x,z) \right) \wedge \left(\left(\underline{R} \right)_{<k>}(z,y) \right)$

$$\leq \left(\underline{R} \right)_{<k>}(x,y) \qquad (7.3)$$

for every k and every (x, y). On the other hand, by the preliminary lemma 7.1 :

$$(\underline{R})_{<k>} = \underline{\quad '(R)_{<k>}' \quad}$$

Hence (7.3) can be rewritten :

$$\bigvee_{z \in E} \left(\underline{\quad '(R)_{<k>}' \quad}(x,z) \right) \wedge \left(\underline{\quad '(R)_{<k>}' \quad}(z,y) \right)$$
$$\leq \underline{\quad '(R)_{<k>}' \quad}(x,y) .$$

It follows that for every k, $'(R)_{<k>}'$ is a transitive mvr, so that $(R)_{<k>}$ is a common transitive relation. This is a necessary and sufficient condition for R to be transitive.

Corollary 11.1 : A quasi-relation ' ρ ' is transitive if and only if ρ is (commonly) transitive.

Proof : Obvious, since $R_{<k>} = \rho$ for every k ♦ Hence, multivalued transitivity turns out to be a proper extension of common transtitivity.

§ 7. 2 *Transitive closure.*

Theorem 12 : Let $(R_i)\ i \in I$ be any family of transitive multivalued relations on E. Then the infimum $R = \bigwedge_{i \in I} R_i$ is transitive.

Proof : On account of the formula (6.7), and the transitivity of R_i, one has : $R^2 =$

$$\left(\bigwedge_{i \in I} R_i \right)^2 = \left(\bigwedge_{i \in I} R_i \right) \otimes \left(\bigwedge_{i \in I} R_i \right) \leq \bigwedge_{i,j \in I^2} (R_i \otimes R_j)$$
$$= \left[\bigwedge_{i \in I} {R_i}^2 \right] \wedge \left[\bigwedge_{\substack{i,j \in I^2 \\ i \neq j}} (R_i \otimes R_j) \right]$$

$$\leq \left[\bigwedge_{i \in I} R_i \right] \wedge \left[\bigwedge_{\substack{i,j \in I^2 \\ i \neq j}} (R_i \otimes R_j) \right] = R \wedge Z \leq R,$$

so that R is transitive.

Corollary 12.1 : For every mvr S there exists the least transitive mvr Ś dominating S, called the *transitive closure* of S.

Proof : Let V be the set of all multivalued transitive relations R on E such that $S \leq R$; V is obviously not empty since it contains $1 = ' E^2 '$. From the previous theorem, it follows that the infimum Ś of V exists and is a transitive relation. Moreover, $Ś \in V$ and $S \leq Ś$, hence Ś is the required multirelation which dominates S.

8. Multivalued preorder relations.

Definition 8.1 : A multivalued relation which is reflexive and transitive is called a *multivalued preorder* (shortly an *mv-preorder*)

Theorem 13 : Let R be a multivalued preorder. Then $R^2 = R$.

Proof : R is transitive so that $R \otimes R \leq R$. And R is reflexive, so that $I \leq R$. It follows $R \otimes I \leq R \otimes R$ (see Theorem 8). Since $R \otimes I = R$ (see Theorem 10), it follows $R \otimes R = R$ ♦

Theorem 14 : A multivalued relation R is a multivalued preorder if and only if every $R_{<j>}$ is a (common) preorder.

Proof : R is an mv-preorder if and only if R is reflexive and transitive. On the one hand, R is transitive if and only if every $R_{<j>}$ is (commonly) transitive (Theorem 11). On the other hand, R is reflexive if and only if every $R_{<j>}$ is (commonly) reflexive (see Remarks 6.2). So that R is an mv-preorder iff. every $R_{<j>}$ is a (common) preorder ♦

It follows that a quasi-relation ' ρ ' is a multivalued preorder if and only if ρ is a (common) preorder relation (proof is obvious since $R_{<k>} = \rho$ for every k). Hence, multivalued preorders turn out to be proper extensions of common ones.

9. Classes of antisymmetry.

Definition 9.1 : An r-valued relation R is said to be *class* k *-antisymmetric* $(1 \le k \le r - 1)$ if its monotonic component $R_{<k>}$ is (commonly) antisymmetric. A mvr R is said to be *antisymmetric* if there exists k $(1 \le k \le r - 1)$ such that R is class k -antisymmetric.

Remarks 9.1 : Recall $R_{<k>}$ is (commonly) antisymmetric if and only if
$(\forall (x, y) \in E^2)\ (x, y) \in R_{<k>}$ and $(y, x) \in R_{<k>} \Rightarrow x = y$. On the other hand, in order that (x, y) belongs to $R_{<k>} = \bigcup_{k \le j \le r-1} R^{<j>}$, it is necessary and sufficient that $\underline{R}(x, y) \ge a_k$, that is, the degree of comparability of x to y is greater or equal than k. It follows that a mvr R is class k-antisymmetric if and only if, whenever both comparabilities, of x to y and y to x, are greater or equal than k, then x = y. Examples : **a)** R is class 1-antisymmetric if and only if $\underline{R}(x, y) \ne 0$ and $\underline{R}(y, x) \ne 0$ imply x = y : in other words, there are not distinct elements such that their mutual degrees of comparability are both non zero. **b)** R is class (r - 1)-antisymmetric if and only if $\underline{R}(x, y) = 1$ and $\underline{R}(y, x) = 1$ imply x = y. In other words, there are not distinct elements such that their mutual degrees of comparability are both maximum. As we can infer from these two examples, antisymmetry may bring some limits to the possibility of differentiating elements.

Theorem 15 : A multivalued relation R is antisymmetric if and only if it is class (r-1)-antisymmetric. A quasi-relation ' ρ ' is antisymmetric if and only if ρ is (commonly) antisymmetric.
Proof : If R is class (r-1)-antisymmetric, then it is antisymmetric. Conversely, assume R to be class k -antisymmetric. Recall this means : $(\forall (x, y) \in E^2)$ $[(x, y) \in R_{<k>}$ and $(y, x) \in R_{<k>} \Rightarrow x = y]$.

Since $R_{<1>} \supset R_{<2>} \supset \ldots \supset R_{<r-1>}$, it follows that, for every $p \ge k$, $(x, y) \in R_{<p>}$ and $(y, x) \in R_{<p>} \Rightarrow (x, y) \in R_{<k>}$ and $(y, x) \in R_{<k>}$ so that x = y. It follows that, if R is class k- antisymmetric, then for every $p \ge k$, R is class p-antisymmetric. We will say that antisymmetry is **"hereditary"** property. Hence every antisymmetric relation is class (r-1)-antisymmetric. If R is the quasi-relation ' ρ ', then $R_{<k>} = \rho$ for every k, so that R is antisymmetric if and only if ρ is (commonly) antisymmetric◆ Hence, multivalued antisymmetry turns out to be a proper extension of common one.

10. Multivalued order relations.

Definition 10.1 : A multivalued relation which is reflexive, transitive and class k- antisymmetric $(1 \le k \le r - 1)$ is called a *class* k-*multivalued order*. A multivalued relation R is said to be a *multivalued order* if there exists k $(1 \le k \le r - 1)$ such that R is a class k-multivalued order. In such cases, for every (x, y) in E^2, the degree of comparability of x to y will be called the *degree of order* of x to y (*w.r.t* R).

Theorem 16 : A multivalued relation R is a class k-multivalued order if and only if R is a multivalued preorder and if $R_{<k>}$ is a (common) order.

Proof : From Definition 10.1, it follows that R is an mv- order of the class k, if and only if two conditions are fulfilled : a) $R_{<k>}$ is (commonly) antisymmetric ; b) R is an mv- preorder. In such case, every $R_{<j>}$ is a common preorder (Theorem 14), so that $R_{<k>}$ is a (common) order ◆ The converse is obviously true.

Theorem 17 : A multivalued relation R is a multivalued order if and only if it is a class (r-1)-multivalued order. A quasi-relation 'ρ' is a multivalued order if and only if ρ is a (common) order.
Proof : If R is a class (r-1)-multivalued order relation, then R is a (multivalued) order. Conversely, if R is a multivalued order, then R is a mv-preorder. Moreover, R must be class k-antisymmetric, for some integer k $(1 \le k \le r-1)$. From Theorem 15, it then follows that R is class (r-1)-antisymmetric, so that R is a class (r-1)-multivalued order ◆If R is the quasi-relation 'ρ ', then all its monotonic components are equal : $R_{<j>} = \rho$ for every j. Hence R = 'ρ ' is a mv-order if and only if ρ is a common order ◆Hence, multivalued orders turn out to be proper extensions of common ones◆Hence a class k-multivalued order R is completely determinated by a decreasing sequence of (r-1) common preorders on E:
$R_{<1>} \supset R_{<2>} \supset \ldots \supset R_{<r-1>}$ such that for $p \ge k$, all the $R_{<p>}$'s are (common) orders. Note that, from the fact that all monotonic components $R_{<p>}$ $(p \ge k)$ are orders in the common sense, *we cannot derive* that the disjunctive components $R^{<p>}$ are (common) orders, except for the case k = r-1, since $R_{<r-1>} = R^{<r-1>}$.

11. Classes of symmetry.

§ 11.1.- Recall that a common relation ρ on E is said to be **symmetric** if $(\forall (x, y) \in E^2)$

$$(x, y) \in \rho \Rightarrow (y, x) \in \rho \qquad (11.1)$$

§11.2. **Definition 11.2** : An r-valued relation R is said to be *class* k *-symmetric* $(1 \le k \le r - 1)$ if its monotonic component $R_{<k>}$ is (commonly) symmetric. A multivalued relation R is said to be *symmetric* if there exists k $(1 \le k \le r - 1)$ such that R is class k -symmetric.

Remarks 11.2 : $R_{<k>}$ is (commonly) symmetric if and only if $(\forall (x, y) \in E^2)$ $(x, y) \in R_{<k>} \Leftrightarrow (y, x) \in R_{<k>}$. On the other hand, in order that (x, y) belongs to $R_{<k>} = \bigcup_{k \le j \le r-1} R^{<j>}$, it is necessary and sufficient that $\underline{R}(x, y) \ge a_k$, that is, the degree of comparability of x to y is greater or equal than k. It follows that a mvr R is class k-symmetric if and only if, for every $(x, y) \in E^2$, $\underline{R}(x, y) \ge a_k$ holds if and only if $\underline{R}(y, x) \ge a_k$ holds. In other words a mvr R is class k-symmetric if and only if, for every $(x, y) \in E^2$, the two mutual degrees of comparability, of x to y and y to x, are both greater or equal than a_k, or alternatively, both lesser than a_{k-1} (in the latter case, we must suppose $k \ge 2$). Examples : **a)** R is class 1-symmetric if and only if $\underline{R}(x, y) \ne a_0 = 0$ holds if and only if $\underline{R}(y, x) \ne 0$ holds. **b)** R is class (r - 1)-symmetric if and only if $\underline{R}(x, y) = a_{r-1} = 1$ holds if and only if $\underline{R}(y, x) = 1$ holds ◆ Moreover it should be emphasized that, while antisymmetry is hereditary (see Theorem 15), symmetry is not. More precisely, assume $R_{<k>}$ to be symmetric and $p \ge k$. Then, $(x, y) \in R_{<p>} \Rightarrow (y, x) \in R_{<k>}$, so that $(y, x) \in R_{<k>}$. From this, we cannot derive $(y, x) \in R_{<p>}$, which would have been necessary to prove the symmetry of $R_{<p>}$.

Theorem 18 : A quasi-relation ' ρ ' is symmetric if and only if ρ is (commonly) symmetric.

The proof is obvious◆Hence, multivalued symmetry turns out to be a proper extension of common symmetry.

§ 11.3.- **Definition 11.3** : An r-valued relation R is said to be *class* k-*strongly symmetric* $(1 \le k \le r - 1)$ if, for every $p \ge k$, all its monotonic components $R_{<p>}$ are (commonly) symmetric. A multivalued relation R is said to be *strongly symmetric* if there exists k $(1 \le k \le r - 1)$ such that R is class k-strongly symmetric.

Clearly every strong symmetric relation (resp. class k-strong symmetric) is symmetric (resp. class k-strongly symmetric), but all converses are false.

Theorem 19 : A multivalued relation R is strongly symmetric if and only if it is class (r-1)-strongly symmetric. A quasi-relation ' ρ ' is strongly symmetric if and only if ρ is (commonly) symmetric.

Proof : If R is class (r-1)-strongly symmetric, then it is strongly symmetric. Conversely, assume R to be class k-strongly symmetric, then for every $p \ge k$, all its monotonic components $R_{<p>}$ are (commonly) symmetric. Hence $R_{<r-1>}$ is symmetric, so that R is class (r-1)-strongly symmetric ◆ If R is the quasi-relation ' ρ ', then $R_{<k>} = \rho$ for every k. Then one has the following logical equivalences : R is mv-symmetric $\Leftrightarrow$ ρ is commonly symmetric $\Leftrightarrow$ R is mv-strongly symmetric.

12. Multivalued equivalence relations.

Definition 12.1 : A multivalued relation R which is reflexive, transitive, and symmetric (resp. class k-symmetric), is called a *multivalued equivalence relation* (resp. a *class* k-*multivalued equivalence relation*). In such cases, for every (x, y) in E^2, the degree of comparability of x to y will be called the *degree of equivalence of* x *to* y (*w.r.t* R).

Definition 12.2 : A multivalued relation which is reflexive, transitive, and strongly symmetric (resp. class k-strongly symmetric), is called a *multivalued strong equivalence relation*, (resp. a *class* k-*multivalued strong equivalence relation*).

Remarks 12.2 : Recall that a mvr R is said to be strongly symmetric if there exists k such that for every $p \ge k$, every $R_{<p>}$ is a (common) symmetric relation, that is to say : $(\forall (x, y) \in E^2)$ $(x, y) \in R_{<p>} \Rightarrow (y, x) \in R_{<p>}$; it follows that every multivalued strong equivalence relation is a multivalued equivalence relation.

Theorem 20 : A multivalued relation R is a multivalued equivalence relation if and only if every $R_{<j>}$ is a (common) preorder and if there exists k $(1 \le k \le r - 1)$ such that $R_{<k>}$ is a (common) order. A multivalued relation R is a multivalued strong equivalence if and only if every $R_{<j>}$ is a (common) preorder and if there exists k $(1 \le k \le r - 1)$ such that for every $p \ge k$, $R_{<p>}$ is a (common) equivalence relation.

The proof is pure routine ◆ Hence a multivalued strong equivalence R is completely determinated by a decreasing sequence of (r-1) common preorders relations on E : $R_{<1>} \supset R_{<2>} \supset \ldots \supset R_{<r-1>}$ such that there exists k such that, for $p \ge k$, all $R_{<p>}$'s are common equivalences.

REFERENCES

[1] DWINGER Ph., «A survey of the theory of Post algebras and their generalizations», in *Modern Uses of Multiple-Valued Logic* (ISMVL 5), J.M Dunn and G. Epstein eds., Reidel, Dordrecht-Boston, 1977, 51-75.

[2] EPSTEIN G., «The lattice theory of Post algebras», *Trans. Amer. Math. Soc.*, **9 5**,(1960), 300-317.

[3] FRAïSSE, R, *Theory of Relations*, North Holland. Amsterdam. New-York. Oxford. 1986.

[4] SERFATI M.,(1996) «On postian algebraic equations», *Discrete Mathematics*, **152,** (1996) 269-285.

[5] SERFATI M. (1997), «A note on postian matrix theory», *International Journal of Algebra and Computation*, **7**(2), (1997), 161-179.

[6] SERFATI M. (1999 a) «The lattice theory of r-ordered partitions», *Discrete Mathematics*, **1 9 4** (1999), 205-227.

[7] SERFATI,M (1999 b) «Quasi-ensembles d'ordre r et approximations de répartitions ordonnées», to appear (1999) in Mathématiques, Informatique, et Sciences humaines.

[8] TRACZYK,T «Axioms and some properties of Post algebras», Colloquium Mathematicum,**1 0**, (1963), 193-209.

Quaternion Groups versus Dyadic Groups in Representations and Processing of Switching Functions

Radomir S. Stanković, Dejan Milenović, Dragan Janković
Dept. of Computer Science
Faculty of Electronics
18 000 Niš,
Yugoslavia

Abstract

In this paper we compare effects of two different domain groups for switching functions to the efficiency of calculation of spectral transforms (ST) representations and the complexity of Decision diagrams (DDs) representations. Dyadic groups and quaternion groups are assumed for domain groups for switching functions. We compared space and time complexity in calculation of STs representations through FFT. DDs are compared in terms of their basic characteristics, the depth, the width, and the size. The area of a DD, defined as the product of size and width, and for word-level DDs, the ratio between the number of non-terminal and constant nodes are discussed as another characteristics of DDs dependent on the domain groups. It is shown that the quaternion groups have advantages in processing of large switching functions. When number of variables grows, these advantages increases.

1. Introduction

An algebraic structure is suitable for logic design if it

1. Provides compact representations of switching functions,
2. Permits computation of the representations for a given function f efficiently in terms of space and time.

The first requirement should ensure realizations with logic networks occupying small area and consisting of a small number of logic elements. Efficiency in computation of a ST-representation is usually a basis for efficient algorithms for manipulation and calculations with represented functions. The same applies to DDs that are graphical representations of ST-representations [16], [17]. Thus, the second requirement strongly interferes with applications, as for example, testability, verification, and similar, performed through these representations.

1.1 Boolean structures

Classically, logic design is performed in the Boolean algebra over the set $B = \{0, 1\}$ in terms of the logic AND ($\vee$) and OR ($\wedge$). The Boolean ring over B in terms of AND and modulo 2 addition, EXOR ($\oplus$), is a closely related algebraic structure.

From 1990, convenience of the Boolean ring in the above sense is greatly confirmed both theoretically and experimentally [8], [9], [10]. Thus, the importance of AND-EXOR synthesis considerably raises [8], [9]. It is further supported by AND-EXOR related decision diagrams [9], which extend its applicability to switching functions with a large number of variables.

1.2 Vector spaces

Spectral transform approach to calculation of polynomial representations in terms of AND and EXOR proved very efficient [1], [2], [3]. In this approach, the switching functions of a given number of variables are considered as elements of linear vector spaces over the Galois field $GF(2)$. The spectral techniques based on other spectral transforms used in switching theory and logic design, consider switching functions as elements of linear vector spaces over the field of rational numbers Q or the complex-field C. It is assumed that the logic values 0 and 1 are formally replaced by the integers 0 and 1, or in some cases suitably coded as $(0, 1) \rightarrow (1, -1)$ [6].

1.3 Domain groups

The group-theoretic approach to switching theory permits to determine a relationship between the Boolean ring and vector spaces used in switching theory. In this approach, the Boolean ring and vector spaces of switching functions over $GF(2)$ or C can be uniformly considered as algebraic structures on finite dyadic groups $C_2^n = C_2 \times \cdots \times C_2$, where $\times$ denotes the direct product, and $C_2 = (B, \oplus)$. This relationship extends to the Boolean algebra through the relation between the Boolean ring and the Boolean algebra. Therefore, the finite dyadic group is a basic algebraic structure usually used for switching theory and logic design.

Definition 1: An n-variable switching function is defined as the mapping $f : C_2^n \rightarrow B$.

Thus, being dependent on two-valued variables, the switching functions are naturally considered as functions on the dyadic groups. At the same time, being two-valued functions, the set of all switching functions of n-variables is the dyadic group $C_2^{2^n}$ under the componentwise EXOR performed over truth-vectors.

These basic features determine complexity of representations, manipulations, and calculations with switching functions. If n is large, these features become a restrictive factor for many methods and techniques. For example, determination of coefficients in ST-representations, as for example, the Reed-Muller expressions and related Kronecker expressions [9], by FFT-like algorithms [1], [2], expresses exponential complexity of $O(2^n)$ and $O(n2^n)$ in terms of space

0-7695-0161-3/99 $10.00 © 1999 IEEE

and time. The same applies to arithmetic, Walsh, and other related representations of switching functions [18]. Thus, if n is large, for example $n > 30$, such algorithms are hardly applicable on most of standard computer architectures.

Decision diagrams on C_2^n are subjected to the same restrictions. Irrespective of the optimization by choosing different decomposition rules, a DD on C_2^n has n levels consisting of nodes with two outgoing edges. Complexity of DDs on C_2^n, as for example Binary DDs (BDDs), approximates on the average $O(2^n/n)$ [9]. Thus, when n is large, we cannot build BDD for f.

In [19], the following question is asked: "The ultimate purpose must be to find out whether this group (the quaternion Q_2) may be as significant for logic synthesis as it seems to be for filtering and other signal processing tasks".

Considerations in this paper are an attempt towards approaching to an answer to this question. Recently, few papers discussed different topics in Fourier analysis on non-Abelian groups and application in switching theory [13], [14], [15], [19]. In this paper, we present a short compilation of some selected results concerning quaternion groups. Then, we give a critical analysis and comparison of them, which shows suitability of quaternion groups for some tasks in logic design. Since there are no proper analogies of the Boolean algebra or Boolean ring based on the quaternion groups, the comparison is restricted to vector space structures based on the dyadic groups and the quaternion groups over C.

1.4 Comparisons

We compare efficiency of dyadic groups and the quaternion groups in

1. Complexity of calculation of coefficients in ST-representations through FFT.
2. Complexity of various DDs on dyadic groups and Fourier DDs on the quaternion groups.

We assume that in an n-variable switching function f, each triplet of variables x_i, x_j, x_k, where each variable takes values in C_2, is replaced by a variable X_r in Q_2. In that way, the domain group C_2^n for f is replaced by Q_2^k, for $n = 3k$, by $C_2Q_2^k$ for $n = 3k+1$, and by $C_4Q_2^k$ for $n = 3k+2$. C_4 is the cyclic group of order 4.

2. Fourier Transform

In this section, we briefly review basic definitions of Fourier transform on non-Abelian groups. For more information, we refer to classical references, as [5], or the more recent publications as [18].

Denote by $C(G)$ the space of complex functions on a finite not necessarily Abelian group G of order g. Denote by Γ the dual object of G. Γ consists of K unitary irreducible representations $\mathbf{R}_w(x)$ of G over C. Note that $\mathbf{R}_w(x)$ stands for a non-singular $(r_w \times r_w)$ matrix over C. For each non-Abelian group, at least one of the representations is of the order $r_w > 1$.

Definition 2: The direct and inverse Fourier transforms of a function $f \in C(G)$ are defined respectively by,

$$\mathbf{S}_f(w) = r_w g^{-1} \sum_{u=0}^{g-1} f(u)\mathbf{R}_w(u^{-1}), \tag{1}$$

$$f(x) = \sum_{w=0}^{K-1} Tr(\mathbf{S}_f(w)\mathbf{R}_w(x)), \tag{2}$$

where for a matrix $\mathbf{Q}$, $Tr(\mathbf{Q})$ denotes the trace of $\mathbf{Q}$, i.e., the sum of elements on the main diagonal of $\mathbf{Q}$.

From this definition, the Fourier coefficients (1) for f on a non-Abelian group are $(r_w \times r_w)$ matrices when $r_w > 1$. The same definition applies to Abelian groups. In that case, all the unitary irreducible representations are one-dimensional, i.e., they reduce to the group characters.

FFT algorithms on non-Abelian groups are introduced in [7]. Algorithms for calculations in this paper are based on their matrix interpretation in [12], [13].

3. Complexity of FFTs

In this section we compare the space and time complexity of FFT on dyadic and quaternion groups by the way of examples of *mcnc* benchmark functions used in logic design and random generated switching functions. Multiple-output functions $f_0 * f_1 * \cdots * f_{q-1}$ are represented by the integer equivalents f_Z determined by the mapping $f_Z = \sum_{i=0}^{q-1} 2^i f_i$ [6]. For hardware limitations, the comparison is restricted to small benchmark functions. For detailed theoretical considerations and further examples we refer to [15].

Walsh transform is the Fourier transform on dyadic groups. Calculation of the Walsh transform of switching function by FFT requires more operations than calculation of other ST-representations on dyadic groups, as for example, the Reed-Muller, Kronecker, or arithmetic expressions [11]. Thus, it is enough to compare complexity of calculations of Fourier transforms on dyadic and quaternion groups. The derived conclusions hold even stronger for other ST-representations on dyadic groups. It is enough to compare implementations through FFT. In DDs based calculations, we actually implement FFT. For a given f, efficiency is achieved thanks to peculiar properties of f, permitting reduction in DDs. Therefore, for a fair comparison for arbitrary functions, we should refer to calculations through DTs. However, in that case, the number of operations is equal to that in FFT. Thus, it is again enough to compare just FFTs.

Fig. 1 compares the number of operations in FFT on dyadic groups of order 2^{3r} and quaternion groups of order r. It shows that for $n > 10$, the quaternion groups require fewer number of operations. Note that the Walsh transform matrix requires just additions and subtractions, while the Fourier transform matrix on Q_2 involves complex unit $i = \sqrt{-1}$. However, in this case, advantages are taken from the appearance of zero elements in the transform matrix. For analytical expressions of the number of operations and

details of implementation of FFT on the compared groups, see [15]. Further details on the subject are available by the first author.

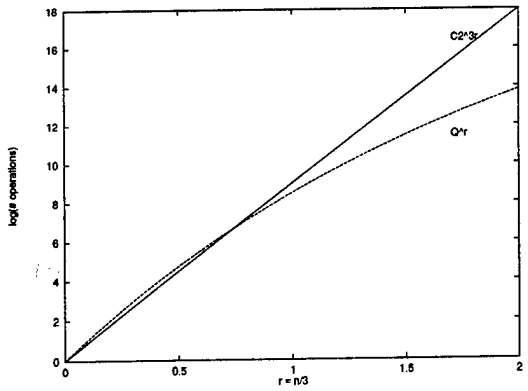


Fig. 1. Number of operations in FFT.

Table I shows time (t) and space (m) required to perform FFT on dyadic (d) and quaternion groups (q) for some switching functions of different number of variables n. For $n = 8, 9, 10, 14$, we use the domain groups C_2^8, C_2^9, C_2^{10}, C_2^{14}, and $C_4Q_2^2$, Q_2^3, $C_2Q_2^3$, $C_4Q_2^4$, respectively. Comparison of the considered groups is given in Table II in terms of the ratio $r_t = t_d/t_q$ and $r_m = m_d/m_q$ of the used time and space. Table III and Table IV give the same information for random generated functions denoted by fun(n) for $n = 9, 10, 12, 14$. Time unit is $1msec = 10^{-3}sec$. The space is given in KBytes. Calculations are performed on a 133MHz Pentium PC with 32MBytes of main memory.

TABLE I
Complexity of FFT.

		d		q	
f	n	t	m	t	m
adr4	8	60.90	46.39	125.80	100.27
misex1	8	33.20	35.14	73.70	89.17
rd84	8	116.10	46.39	133.90	101.39
mul4	8	63.40	45.27	129.70	99.29
9sym	9	379.80	86.39	324.60	193.30
apex4	9	464.00	86.39	336.00	196.96
clip	9	466.00	86.39	339.00	197.38
adr5	10	759.00	166.39	787.00	381.89
mul5	10	788.00	164.14	833.00	379.93
sao2	10	1531.00	166.39	763.00	375.99
adr7	14	233360.00	1926.39	78360.00	5334.39
misex3	14	801480.00	1926.39	75060.00	5331.19
mul7	14	123200.00	1618.28	65320.00	5165.25

Fig. 2 compares time required to perform FFT on C_2^{3r} and Q_2^r for different values of r. Fig. 3 compares the corresponding space requirements.

This consideration shows that quaternion groups permit faster implementation of FFT for large switching functions

TABLE II
Comparisons of domain groups.

f	n	r_t	r_m
adr4	8	2.07	2.16
misex1	8	2.22	2.54
rd84	8	1.15	2.19
mul4	8	2.04	2.19
9sym	9	0.85	2.24
apex4	9	0.72	2.28
clip	9	0.73	2.28
adr5	10	1.04	2.30
mul5	10	1.06	2.31
sao2	10	0.50	2.26
adr7	14	0.34	2.77
misex3	14	0.09	2.77
mul7	14	0.53	3.19

TABLE III
FFT for random functions.

f	t	m	t	m
fun(9)	536.00	86.39	352.00	198.64
fun(10)	2048.00	166.39	916.00	390.89
fun(12)	39620.00	486.39	7110.00	1356.32
fun(14)	949000.00	1930.30	94280.00	5406.07

TABLE IV
Comparison of FFT for random functions.

f	r_t	r_m
fun(9)	0.66	2.30
fun(10)	0.45	2.35
fun(12)	0.18	2.79
fun(14)	0.10	2.80

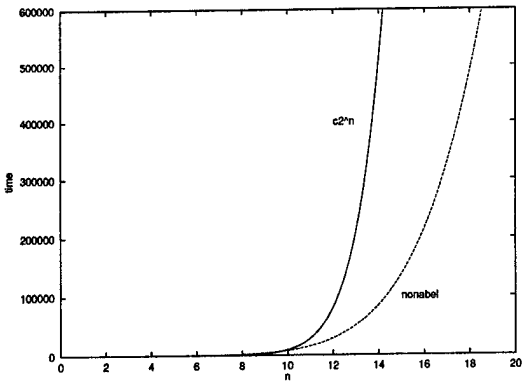


Fig. 2. Time requirements.

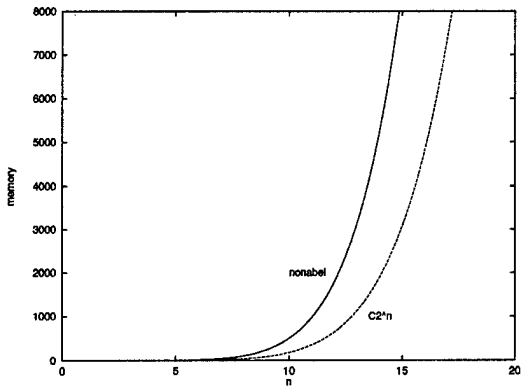


Fig. 3. Memory requirements.

than dyadic groups, at the price of a comparatively small increase of the required space. For example, for $n > 10$ for at about ten times faster implementation of FFT, three times more space is required. When the number of variables n grows, the advantages of quaternion groups increase.

4. Decision Diagrams

Decision Diagrams (DD) are convenient data structure for representation of discrete functions [11]. A DD is a rooted acyclic graph consisting of the root node, a set of non-terminal nodes, and a set of constant nodes connected with edges. For a given function f, a DD is designed by the reduction of the corresponding Decision Tree (DT) representing f [11].

Complexity of a DD is usually characterized by its

1. size - the total number of nodes (non-terminal and constant nodes),
2. width - the maximal number of non-terminal nodes per a level, where a level in a DD consists of nodes to which the same variable in f is assigned.
3. depth - the number of levels in the DD.

We suggest to consider the area (a) of a DD defined as the product of width (w), and size (s), thus $a = ws$. In DDs bases logic design, the area should be a rough measure of space occupied by the networks produced through DDs.

In word-level DDs, it is useful to consider the ratio r_n of non-terminal (ntn) and constant nodes (cn), $r_n = ntn/cn$. In DDs based realizations, each non-terminal node requires a circuit. In DDs based calculations, a subprocedure is assigned to each non-terminal node. The constant nodes are considered as inputs into circuits, respectively, subprocedures. Thus, in some applications, it may be useful to reduce the number of non-terminal nodes at the price of increased number of constant nodes. Clearly, in bit-level DDs, $r_n = s/2$.

4.1 Fourier decision trees

Various DDs are defined by using different decomposition rules [11]. Fourier DDs are defined with respect to the Fourier decomposition defined by (2). From FFT theory, the Fourier transform on a finite group G decomposable into the product of n subgroups G_i may be considered as the n-dimensional Fourier transform on the constituent subgroups G_i [14].

Definition 3: Fourier decision tree on G is defined as the decision tree whose nodes at the i-th level represent functions

$$f(x_i) = \sum_{w_i \in \Gamma_i} Tr(\mathbf{S}_f(w_i)\mathbf{R}_{w_i}(x_i)), \quad x_i \in G_i, \qquad (3)$$

where Γ_i is the dual object of G_i, and $\mathbf{R}_{w_i}$ are the unitary irreducible representations of G_i.

In Fourier DT for a given f, the values of constant nodes are the Fourier coefficients of f. The Fourier coefficients corresponding to the group representations with orders $r_w > 1$, $w = 1, \ldots, K-1$, are $(r_w \times r_w)$ matrices. Therefore, Fourier DTs on finite non-Abelian groups (FNADDs) are matrix-valued DTs, since some of their constant nodes are $(r_w \times r_w)$ matrices. The matrix-valued coefficients may be also represented by the DTs on Abelian groups by using the method of representation of matrices by decision diagrams [3]. In that way, we derive the number-valued FNADDs. In this paper, we consider number-valued FNADDs, since we do comparison with DDs on Abelian groups that are the number-valued DDs.

4.2 Fourier decision diagrams

Fourier decision diagrams are derived by the reduction of Fourier decision trees. The reduction is performed by using generalized BDD reduction rules introduced for the reduction of the Walsh transform DDs (WDDs), which are Fourier DDs on finite dyadic groups [17].

Definition 4: Fourier decision diagrams are DDs derived from the Fourier decision trees by using the generalized BDD reduction rules. A Fourier decision diagram is reduced if further reduction with the same rules is impossible.

5. Complexity of FNADDs

In this section, we compare various DDs on dyadic groups with FNADDs on the quaternion groups. Multiple-output functions are represented by Shared BDDs (SBDDs), Multi-terminal binary DDs (MTBDDs) [11], and FNADDs. For representation by MTBDDs and FNADDs, they are first represented by the integer-valued functions f_Z. Depending on the value of n, in FNADDs on quaternion groups $C_2Q_2^r$ and $C_4Q_2^r$, we use as the root node the Shannon nodes with two and four outgoing edges.

Table V compares sizes and widths of SBDDs and FNADDs for various benchmark functions. For FNADDs the values of non-terminal nodes (ntn) and constant nodes (cn) are shown separately. Thus, the size of FNADDs is the sum of these two values. This table shows that, besides depth reduction, we get the width reduction, except for 5xp1. Except this function and con1, the area is also reduced. In FNADDs, r_n is quite more convenient. For example, in xor5, where the size and width remain the same, it is 5.5 and 0.83 for SBDDs and FNADDs, respectively.

TABLE V
SBDDs and FNADDs for benchmark functions.

	SBDD			FNADD						
f	size	width	a	ntn	cn	size	width	a	r_n	group
5xp1	90	25	2250	39	128	167	18	3006	0.35	$C_2Q_2^2$
bw	116	37	4292	9	25	34	4	136	0.36	C_4Q_2
con1	20	5	100	13	12	25	6	150	1.83	$C_2Q_2^2$
rd53	25	6	150	7	14	21	3	63	0.50	C_4Q_2
rd73	45	10	450	23	30	53	6	318	0.70	$W_2Q_2^2$
xor5	11	2	22	5	6	11	2	22	0.83	C_4Q_2

Table VI compares sizes of SBDDs and FNADDs for n-bit adders. In this example, FNADDs provide the reduction of all three parameters, depth, size, and width and the reduction is considerable compared to SBDDs. The reduction possibilities in these DDs are compared through the percent of used nodes from the total of nodes in the corresponding DTs. It may be seen that these values are comparable, thus, the possibility to do reduction in SBDDs and FNADDs is comparable. In FNADDs, the reduction of area is considerable and shows advantages of them.

Table VII compares the FNADDs to some other DDs based on spectral transforms. The Arithmetic transform DDs (ACDDs), Walsh transform DDs (WDDs) in $(1,-1)$ coding [17], and Complex-Hadamard Transform DDs [4] are compared with FNADDs for some benchmark functions. ACDDs, WDDs, and CHTDDs are DDs on Abelian groups. Therefore, the depth is equal to n. We do not use peculiar properties of complex spectra for switching functions in CHTDDs, nor FNADDs to further reduce the depth of these DTs. For rd53 and xor5, the sizes of FNADDs are equal to those of other DDs. However, the number of non-terminal nodes and the width are reduced. In other cases in this example, FNADDs are more efficient with respect to depth, width and size. However, as for other DDs, some examples where FNADDs are not efficient certainly can be found. As an example consider the Achille's function $f = x_1y_1 \vee x_2y_2 \vee \cdots \vee x_{2r}y_{2r}$. Table VIII shows the sizes of FNADDs for Achille's heal function for different values of n and for two different orderings of variables $x_1, x_2, \ldots, x_{2r}, y_1, y_2, \ldots, y_{2r}$ and $x_1, y_1, x_2, y_2, \ldots, x_n y_n$. This example shows that, as in any other DDs, the sizes of FNADDs greatly depends on the initial variables ordering. In this example, the depth reduction in FNADDs is achieved at the price of the increased size of FNADDs.

6. Closing Remarks

For large switching functions,

1. Quaternion groups permit considerable reduction of time complexity of FFT at the price of comparatively small increase of space complexity.
2. FNADDs on quaternion groups permit depth and width reduction simultaneously. In many cases, they have smaller size than DDs on dyadic groups. FNADDs have smaller area and permit reduction of the number of non-terminal nodes at the price of increased number of constant nodes.
3. Both properties of quaternion groups increase when the number of variables, equivalently, the order of domain groups increases.

TABLE VIII
BDDs and FNADDs for Achilles' heal functions.

$n = 2r$	BDD		FNADD		
	worst	best	worst	best	decomposition
4	8	6	13	12	C_2Q_2
6	12	8	37	25	Q_2^2
8	14	10	71	40	$C_4Q_2^2$

It follows that, compared to dyadic groups, the quaternion groups have advantages as domain groups for large switching functions. Extensions to multiple-valued functions, are straightforward. For example, the quaternary functions are defined on C_4^n, or alternatively on the quaternion groups of the order 4^n. However, unlike Walsh transform on C_2, the Fourier transform matrix on C_4 involves complex unit i and unlike the Fourier transform matrix on Q_2 does not have any zero element. Therefore, compared to C_2^n, advantages of Fourier transform on quaternion groups over Fourier transform on C_4^n increase.

References

[1] Ph.W. Besslich, "Efficient computer method for XOR logic design", *IEE Proc., Part E*, Vol.129, 1982, 15-20.

[2] Ph.W. Besslich, "Spectral processing of switching functions using signal flow transformations", in: Karpovsky, M.G., (Ed.), *Spectral Techniques and Fault Detection*, Academic Press, Orlando, Florida, 1985.

[3] E.M. Clarke, K.L. McMillan, X. Zhao, M. Fujita, "Spectral transforms for extremely large Boolean functions", in: Kebschull, U., Schubert, E., Rosenstiel, W., Eds., *Proc. IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, 16-17.9.1993, Hamburg, Germany, 86-90.

[4] B.J. Falkowski, S. Rahardja, "Complex spectral decision diagrams", *Proc. 26th Int. Symp. on Multiple-Valued Logic*, Santiago de Compostela, Spain, May 1996, 255-260.

[5] E. Hewitt, K.A. Ross, *Abstract Harmonic Analysis*, I,II, Springer-Verlag, Berlin, 1963, 1970.

TABLE VI
SBDDs and FNADDs for adders and multipliers.

	adders										
	SBDD				FNADD						
n	size	width	%	a	ntn	cn	size	width	%	a	r_n
2	8	21	22.00	168	4	7	11	2	52.38	22	0.57
3	57	20	42.86	1140	6	7	13	4	31.70	52	0.85
4	103	30	19.84	3090	14	14	28	7	33.73	196	1.00
5	226	62	10.98	14012	18	16	34	7	8.31	238	1.12
6	477	126	5.81	60102	21	12	33	7	4.00	231	1.75
	multipliers										
	SBDD				FNADD						
n	size	width	%	a	ntn	cn	size	width	%	a	r_n
2	19	5	14.28	95	4	10	14	2	9.52	28	0.40
3	63	15	11.27	945	9	20	29	4	9.75	116	0.45
4	159	39	7.51	6201	24	42	66	11	13.25	726	0.57
5	473	114	5.54	53912	37	50	87	17	4.51	1479 7	0.74
6	788	192	2.34	151296	45	49	94	22	2.68	2068	0.92

TABLE VII
Sizes of ACDDs, WDDs, CHTDDs and FNADDs.

	ACDD						CHTDDs					
f	ntn	cn	size	width	a	r_n	ntn	cn	size	width	a	r_n
5xp1	38	11	49	10	490	3.45	127	128	255	64	16320	0.99
bw	31	32	63	16	1008	0.96	31	32	63	16	1008	0.96
con1	40	5	45	10	450	8	115	42	157	56	8792	2.74
rd53	15	6	21	5	105	2.5	15	6	21	5	105	2.5
rd73	27	6	33	6	188	4.5	27	8	36	7	252	3.37
xor5	15	6	21	5	105	2.5	9	2	11	2	22	4.5
	WDD(1,-1)						FNADD					
f	ntn	cn	size	width	a	r_n	ntn	cn	size	width	a	r_n
5xp1	37	13	50	9	450	2.85	39	128	167	18	3006	0.30
bw	31	32	63	16	1008	0.97	9	25	34	4	136	0.36
con1	48	11	59	13	767	4.36	13	12	25	6	150	1.08
rd53	15	5	20	5	100	3.00	7	14	21	3	63	0.50
rd73	28	5	33	7	231	5.6	23	30	53	6	318	0.77
xor5	5	2	7	1	7	2.50	5	6	11	2	22	0.83

[6] M.G. Karpovsky, *Finite Orthogonal Series in the Design of Digital Devices*, John Wiley and JUP, 1976.

[7] M.G. Karpovsky, "Fast Fourier transforms on finite non-Abelian groups", *IEEE Trans. Comput.*, Vol. C-26, No. 10, 1977, 1028-1030.

[8] T. Sasao, *Logic Design: Switching Circuit Theory*, Kindai Kagaku, Japan, second edition, 1998.

[9] T. Sasao, "Representations of logic functions by using EXOR operators", in: [11], 29-54.

[10] T. Sasao, Ph.W. Besslich, "On the complexity of MOD-2 sum PLA's", *IEEE Trans. Comput.*, Vol.33, No.2, 1990, 262-266.

[11] T. Sasao, M. Fujita, (Eds.), *Representations of Discrete Functions*, Kluwer Academic Publishers, 1996.

[12] R.S. Stanković, "Matrix-interpretation of fast Fourier transforms on finite non-Abelian groups', *Proc. Int. Conf. on Signal Processing, Beijing/90*, October 22-26, 1990, Beijing, China, 1187-1190.

[13] R.S. Stanković, "Fast Fourier transform on finite non-Abelian groups", in [18], 405-420.

[14] R.S. Stanković, "Fourier decision diagrams for optimization of decision diagrams representations of discrete functions", *Proc. Workshop on Post Binary-Ultra Large Scale Integration*, Santiago de Campostela, Spain, 1996, 8-12.

[15] R.S. Stanković, D. Milenović, "Some remarks on calculation complexity of Fourier transforms on finite groups", *Proc. 14th European Meeting on Cybernetics and Systems Research, CSMR'98*, April 15-17, 1998, Vienna, Austria, 59-64.

[16] R.S. Stanković, T. Sasao, "Decision diagrams for discrete functions: Classification and unified interpretation", *ASP-DAC'98*, February 13-17, 1998, 349-446.

[17] R.S. Stanković, T. Sasao, C. Moraga, "Spectral transform decision diagrams" in: [11], 55-92.

[18] R.S. Stanković, M.R. Stojić, M.S. Stanković, *Recent Developments in Abstract Harmonic Analysis with Applications in Signal Processing*, Nauka, Belgrade and Elektronski fakultet, Niš, 1996.

[19] E.A. Trachtenberg, "Applications of Fourier Analysis on Groups in Engineering Practices", in: [18], 331-403.

On Axiomatization of Conditional Entropy of Functions Between Finite Sets

Szymon Jaroszewicz
Univ. of Massachusetts
at Boston, Dept. of
Math. and Comp. Science,
Boston, 02125 USA

Dan A. Simovici
Univ. of Massachusetts
at Boston, Dept. of
Math. and Comp. Science,
Boston, 02125 USA

Abstract

In this paper we present a new axiomatization of the notion of entropy of functions between finite sets and we introduce and axiomatize the notion of conditional entropy between functions. The results can be directly applied to logic functions, which can be regarded as functions between finite sets. Our axiomatizations are based on properties of entropy with regard to operations commonly applied to discrete functions and are related to the usage of entropy as a measure of the energy dissipated by circuits that implement discrete functions.

Keywords: functions between finite sets, logic functions, entropy, axiomatization, conditional entropy.

1 Introduction and Basic Notations

This paper is concerned with the axiomatization of the notion of entropy for finite functions. This type of numerical characteristic of functions has been shown (cf.[1]) to be related to the complexity of realization of boolean functions. It has also been noted [3], that power dissipation of a circuit realizing a binary function is proportional to its entropy. Information measures, especially conditional entropy of a logic function and its variables, have been used for minimization of logic functions [4]. A study of information estimations for logic functions can be found in [2]. Axiomatizations of entropies for random variables can be found for example in [5]. The axiomatization of Shannon entropy for functions considered in this paper makes use of the operations that are naturally encountered in building circuits (parallel and serial connections of functional modules) and represents a simplification of some of our previous work [6]. Further, we extend the notion of conditional entropy of a function and its arguments to conditional entropy between functions and we give an axiomatization of this notion.

Unless otherwise specified, all sets considered in the following discussion are nonempty and finite. All logarithms are in base 2. In the discussion we assume that $0 \cdot \log 0 = 0$. The domain and range of a function f are denoted by Dom (f) and Ran(f) respectively. The restriction of function f to a set $A \subseteq$ Dom (f) is denoted by f_A. Below we introduce some operations on functions between finite sets.

Let $\mathcal{F}$ be the class of all functions between finite, nonempty sets. Define the partial order $\sqsubseteq$ on $\mathcal{F}$ by $f \sqsubseteq g$ if $f : A \longrightarrow B$, $g : A' \longrightarrow B'$, $A \subseteq A'$, $B \subseteq B'$ and $f(a) = g(a)$ for $a \in A$. Note that if A,B,C,D are finite sets, such that $A \cap B = C \cap D = \emptyset$, $f : A \longrightarrow B$, and $g : C \longrightarrow D$, then there exists $\sup\{f, g\}$. We define the function $f \sqcup g = \sup\{f, g\}$ by setting

$$(f \sqcup g)(x) = \begin{cases} f(x) & \text{if } x \in A \\ g(x) & \text{if } x \in B \end{cases}$$

It is easily verifiable, that, whenever it is defined, the "$\sqcup$" operation is commutative and associative.

The composition of functions $f : A \to B$ and $g : B \to C$ is a function $gf : A \to C$, such that $gf(x) = g(f(x))$, for every $x \in A$.

The Cartesian product of functions $f : A \to B$ and $g : C \to D$ is a function $f \times g : A \times C \to B \times D$ defined as $(f \times g)(x, y) = (f(x), g(y))$, for every $x \in A$ and $y \in C$.

In the following sections we will also use the following definition:

Definition 1.1 Let $A_1, A_2, \ldots, A_n$ be n pairwise disjoint subsets of a finite set A. The *generalized characteristic function* of sets $A_1, A_2, \ldots, A_n$ is the function: $\chi_{A_1,A_2,\ldots,A_n} : A_1 \cup A_2 \cup \ldots \cup A_n \to \{1, 2, \ldots, n\}$ defined by

$$\chi_{A_1,A_2,\ldots,A_n}(x) = i$$

0-7695-0161-3/99 $10.00 © 1999 IEEE

if and only if $x \in A_i$, for $1 \leq i \leq n$. □

Note, that the generalized characteristic function is also a function between finite sets.

2 Axiomatic definition of functional entropy

In this section we introduce an axiomatic definition of entropy of a function between finite sets.

Definition 2.1 Let $\mathcal{F}$ be the class of all functions between finite, nonempty sets, and $H : \mathcal{F} \to \mathbb{R}$ be a function assigning a real number to every $f \in \mathcal{F}$. Function H is called *entropy* of functions between finite sets if and only if it satisfies the following axioms:

(E1) $H(f\alpha) = H(f)$, for every function $f : A \to B$ and bijection $\alpha : A' \to A$.

(E2) $H(gf) \leq H(f)$, for every $f : A \to B$ and $g : B \to C$.

(E3) If A, C are finite sets with $|A| \leq |C|$, $\alpha : A \to B$ and $\beta : C \to D$ are bijections, then $H(\alpha) \leq H(\beta)$.

(E4) If $f : A \to B$ and $f : C \to D$ are functions between finite sets and $A \cap C = B \cap D = \emptyset$, then

$$H(f \sqcup g) = \frac{|A|}{|A \cup C|} H(f) + \frac{|C|}{|A \cup C|} H(g) + H(\chi_{A,C}).$$

(E5) $H(f \times g) = H(f) + H(g)$ for all functions $f : A \to B$ and $g : C \to D$.

□

We now prove several consequences of the above axioms.

Lemma 2.2 *If $f : A \to f(A) \subset B$ and $f' : A \to B$, such that $f(x) = f'(x)$ for every $x \in A$, then $H(f) = H(f')$.*

Proof. Let $g_1 : f(A) \to B$ and $g_2 : B \to f(A)$ be functions, such that $g_1(x) = x$, for all $x \in f(A)$ and $g_2(x) = x$, if $x \in f(A)$ and $g_2(x) = a$, for $x \notin f(A)$, where a is any element of $f(A)$. Clearly, $f = g_2 f'$ and $f' = g_1 f$. Using **(E2)** we obtain $H(f) = H(g_2 f') \leq H(f')$ and $H(f') = H(g_1 f) \leq H(f)$, which gives $H(f) = H(f')$. ∎

It follows from the above lemma, that only surjective functions need to be considered. In the discussion below we assume all functions to be surjective, unless otherwise specified.

Lemma 2.3 *If $f : A \to B$ is a function and $\beta : B \to B'$ is a bijection, then $H(\beta f) = H(f)$.*

Proof. Note that $f = \beta^{-1}(\beta f)$. Using **(E2)** we obtain $H(f) = H(\beta^{-1}(\beta f)) \leq H(\beta f)$ and $H(\beta f) \leq H(f)$, which gives $H(\beta f) = H(f)$. ∎

Lemma 2.4 *The entropy of a constant function is* 0.

Proof. Let $f : A \to B$ be a constant function. Choose a set C and a constant function $g : C \to D$, such that $|C| = |A|$ and $A \cap C = B \cap D = \emptyset$. Notice that there exists a bijection γ, such that $f \sqcup g = \gamma \chi_{A,C}$, so $H(f \sqcup g) = H(\chi_{A,C})$. From **(E4)** we have $H(f \sqcup g) = \frac{1}{2}H(f) + \frac{1}{2}H(g) + H(\chi_{A,C})$, which gives $H(f) + H(g) = 0$. But, since $|A| = |C|$ and $|B| = |D| = 1$, there exist bijections α and β, such that $g = \beta f \alpha$. Using **(E1)** and lemma 2.3 we get $H(g) = H(f)$, so $H(f) = 0$. ∎

Lemma 2.5 *The entropy of a bijection depends solely on the cardinality of its domain.*

Proof. Let $f : A \to B$ and $g : C \to D$ be any bijections, such that $|A| = |C|$. Since f and g are bijections, we have $|B| = |D|$. Consequently, there exist two bijections $\alpha : A \to C$ and $\beta : D \to B$ such that $f = \beta g \alpha$. By axiom **(E1)** and Lemma 2.3 we get $H(f) = H(g)$. ∎

Denote by α_n a bijection for which $|\mathrm{Dom}\ (\alpha_n)| = n$. A consequence of the above lemma is the existence of a function $h_N : \mathbb{N} \to \mathbb{R}$ from the set of natural numbers into the set of real numbers, such that $h_N(n) = H(\alpha_n)$.

Lemma 2.6 *Let $f : A \to B$ be a function between two finite sets and let $B = \{b_1, \ldots, b_m\}$. The entropy of the function f is given by*

$$H(f) = h_N(|A|) - \sum_{j=1}^{m} \frac{|A_j|}{|A|} h_N(|A_j|),$$

where $A_j = \{a \in A | f(a) = b_j\} = f^{-1}(b_j)$ for $1 \leq j \leq m$.

Proof. Consider functions $f_1 : C_1 \to D_1$ and $f_2 : C_2 \to D_2$, where $C_1 \cap C_2 = D_1 \cap D_2 = \emptyset$ also, let $f = f_1 \sqcup f_2$ and $f : C \to D$, $C = C_1 \cup C_2$, $D = D_1 \cup D_2$. Denote by $\tilde{f}_1 : C \to D_1 \cup \{d\}$, $d \in D_2$, a function such that

$$\tilde{f}_1(x) = \begin{cases} f_1(x) & \text{if } x \in C_1 \\ d & \text{if } x \in C_2 \end{cases}$$

It follows from **(E4)** that:

$$\begin{aligned} H(\tilde{f}_1) &= \frac{|C_1|}{|C|}H(f_1) + \frac{|C_2|}{|C|}H(f_d) + H(\chi_{C_1,C_2}) \\ &= \frac{|C_1|}{|C|}H(f_1) + H(\chi_{C_1,C_2}), \end{aligned}$$

where f_d is the constant function $f_d(x) = d$ for $x \in C_2$.

Applying **(E4)** to f and substituting the above equation, we obtain:

$$H(f) = H(\tilde{f}_1) + \frac{|C_2|}{|C|}H(f_2). \quad (1)$$

Let $f : A \to B$, $B = \{b_1, \ldots, b_m\}$ and $A_i = \{x \in A | f(x) = b_i\}$. The function f can be decomposed using the "$\sqcup$" operation and constant functions as $f = f_{A_1} \sqcup \cdots \sqcup f_{A_m}$. Also, let $\alpha^{(i)} : A_i \to E_i$ be a bijection onto some set E_i, $E_i \cap B = \emptyset$, and $E_i \cap E_j = \emptyset$ for every $i \neq j$. Consider a sequence of functions $F_0, F_1, \ldots, F_m$ such that

$$F_0 = f, F_j = \alpha^{(1)} \sqcup \ldots \sqcup \alpha^{(j)} \sqcup f_{A_{j+1}} \sqcup \ldots \sqcup f_{A_m}$$

for $j > 0$.

Clearly, $F_m : A \to \bigcup E_i$ is a bijection. By substituting $f_{A_1} \sqcup \cdots \sqcup f_{A_m}$ and $\alpha^{(1)}$ into formula (1) in place of $\tilde{f}_1$ and f_2 respectively, we obtain:

$$H(F_0) = H(F_1) - \frac{|A_1|}{|A|}H(\alpha^{(1)}).$$

The same procedure is then applied iteratively to $H(F_1), H(F_2), \ldots, H(F_m)$. E.g., to decompose $H(F_j)$ we substitute $\alpha^{(1)} \sqcup \cdots \sqcup \alpha^{(j)} \sqcup f_{A_{j+1}} \sqcup \ldots \sqcup f_{A_m}$ and $\alpha^{(j+1)}$ in place of $\tilde{f}_1$ and f_2 respectively, into formula (1). By applying this procedure m times, the following equation is obtained:

$$\begin{aligned} H(f) = H(F_0) &= H(F_m) - \sum_{j=1}^{m} \frac{|A_j|}{|A|}H(\alpha^{(j)}) \\ &= h_N(|A|) - \sum_{j=1}^{m} \frac{|A_j|}{|A|}h_N(|A_j|). \end{aligned}$$

∎

Lemma 2.7 *$h_N(n) = r\log(n)$, where $r \in \mathbb{R}, r \geq 0$ is a constant.*

Proof. Let $\alpha_n^k = \underbrace{\alpha_n \times \cdots \times \alpha_n}_{k \text{ times}}$. Let us prove that $H(\alpha_n^k) = kH(\alpha_n)$, for $k \geq 1$. If $k = 1$ the case is trivial. Suppose, that the hypothesis holds for some $k \geq 1$, we have $H(\alpha_n^{k+1}) = H(\alpha_n^k \times \alpha_n)$, from **(E5)** it follows that $H(\alpha_n^{k+1}) = H(\alpha_n^k) + H(\alpha_n) = (k+1)H(\alpha_n)$, which proves the hypothesis.

Since a Cartesian product of bijections is also a bijection we have $h_N(n^k) = kh_N(n)$. On the other hand, it follows from **(E3)**, that h_N is nondecreasing. It is a functional result that a function $h_N : \mathbb{N} \to \mathbb{R}$, satisfying the two conditions has the form:

$$h_N(n) = r\log(n),$$

where $r \in \mathbb{R}, r \geq 0$ is a constant. A proof can be found, for example, in [5]. ∎

Later on, we will explicitly assume $r = 1$. This could also be achieved by introducing an additional normalization condition $H(\alpha_2) = 1$.

We are now ready to prove the following theorem:

Theorem 2.8 *The entropy of a function $f : A \to B$ between two finite sets, $B = \{b_1, \ldots, b_m\}$ has the form:*

$$H(f) = -\sum_{i=1}^{m} \frac{|A_i|}{|A|} \log \frac{|A_i|}{|A|},$$

where $A_i = \{a \in A | f(a) = b_i\}$.

Proof. By applying Lemma 2.6 to the function f and using Lemma 2.7 we get

$$\begin{aligned} H(f) &= \log|A| - \sum_{i=1}^{m} \frac{|A_i|}{|A|} \log|A_i| \\ &= \left(\sum_{i=1}^{m} \frac{|A_i|}{|A|}\right) \log|A| - \sum_{i=1}^{m} \frac{|A_i|}{|A|} \log|A_i| \\ &= -\sum_{i=1}^{m} \frac{|A_i|}{|A|} \log \frac{|A_i|}{|A|}. \end{aligned}$$

∎

3 Axiomatic definition of conditional entropy

In this section we introduce an axiomatic definition of the conditional entropy between functions.

Definition 3.1 Let $\mathcal{F}$ be the class of all functions between finite, nonempty sets. Denote by $\mathcal{D}$ a class of pairs of functions having identical domains $\mathcal{D} = \{(f, g) \in \mathcal{F} \times \mathcal{F} | \text{Dom}\ (f) = \text{Dom}\ (g)\}$. Let $H_C : \mathcal{D} \to \mathbb{R}$ be a

function assigning a real number to every pair of functions $(f,g) \in \mathcal{D}$. $H_C(f,g)$ will be denoted below as $H(f|g)$. Function $H_C(f,g)$ is called *conditional entropy* of a pair of functions (f,g) if and only if it satisfies the following axioms:

(CE1) $H(f|c) = H(f)$ for every constant function c.

(CE2) $H(f|g_1 \sqcup g_2) = \frac{|A_1|}{|A|}H(f_{A_1}|g_1) + \frac{|A_2|}{|A|}H(f_{A_2}|g_2)$, for all $f : A \to B$, $g_1 : A_1 \to C_1$, $g_2 : A_2 \to C_2$, $A_1 \cap A_2 = C_1 \cap C_2 = \emptyset$, $A_1 \cup A_2 = A$.

□

Theorem 3.2 *Conditional entropy between functions $f : A \to B$ and $g : A \to C$, where $B = \{b_1, \ldots, b_m\}, C = \{c_1, \ldots, c_n\}$, has the following form:*

$$H(f|g) = -\sum_{j=1}^{n}\sum_{i=1}^{m} \frac{|A_j^i|}{|A|} \log \frac{|A_j^i|}{|A_j|},$$

where $A_j^i = \{x \in A | f(x) = b_i \wedge g(x) = c_j\}$, $A_j = \{x \in A | g(x) = c_j\}$.

Proof. We decompose g as $g = g_{A_1} \sqcup \ldots \sqcup g_{A_n}$, where $g_{A_1}, \ldots, g_{A_n}$ are constant functions. Let $A' = A_2 \cup \ldots \cup A_n$. Using **(CE2)** we get

$$H(f|g) = \frac{|A_1|}{|A|}H(f_{A_1}|g_{A_1}) + \frac{|A'|}{|A|}H(f_{A'}|g_{A'})$$

Since g_{A_1} is a constant, using **(CE1)** and Theorem 2.8 we get

$$\begin{aligned} H(f|g) &= \frac{|A_1|}{|A|}H(f_{A_1}) + \frac{|A'|}{|A|}H(f_{A'}|g_{A'}) \\ &= -\frac{|A_1|}{|A|}\sum_{i=1}^{m}\frac{|A_1^i|}{|A_1|}\log\frac{|A_1^i|}{|A_1|} + \frac{|A'|}{|A|}H(f_{A'}|g_{A'}) \\ &= -\sum_{i=1}^{m}\frac{|A_1^i|}{|A|}\log\frac{|A_1^i|}{|A_1|} + \frac{|A'|}{|A|}H(f_{A'}|g_{A'}) \end{aligned}$$

Let $A'' = A_3 \cup \ldots \cup A_n$. Applying the above procedure to $H(f_{A'}|g_{A'})$ yields

$$\begin{aligned} H(f|g) &= -\sum_{i=1}^{m}\frac{|A_1^i|}{|A|}\log\frac{|A_1^i|}{|A_1|} - \sum_{i=1}^{m}\frac{|A_2^i|}{|A|}\log\frac{|A_2^i|}{|A_2|} \\ &\quad + \frac{|A''|}{|A|}H(f_{A''}|g_{A''}) \end{aligned}$$

Repeating further the above process we get

$$H(f|g) = -\sum_{j=1}^{n}\sum_{i=1}^{m}\frac{|A_j^i|}{|A|}\log\frac{|A_j^i|}{|A_j|}.$$

■

We now examine some of the properties of conditional entropy defined above.

Theorem 3.3 *$H(f|f) = 0$, for every function between finite sets $f : A \to B$, $B = \{b_1, \ldots, b_m\}$.*

Proof. From Theorem 3.2 we obtain:

$$H(f|f) = -\sum_{j=1}^{n}\sum_{i=1}^{m}\frac{|A_j^i|}{|A|}\log\frac{|A_j^i|}{|A_j|}, \qquad (2)$$

where $A_j^i = \{x \in A | f(x) = b_i \wedge f(x) = b_j\}$, $A_j = \{x \in A | f(x) = b_j\}$. Since

$$A_j^i = \begin{cases} \emptyset & \text{if } i \neq j \\ A_j & \text{if } i = j, \end{cases}$$

by substituting the above into equation (2) we obtain the desired result. ■

Consider the functions $f : A \to B$ and $g : A \to C$, where $B = \{b_1, \ldots, b_m\}$ and $C = \{c_1, \ldots, c_n\}$. Let us denote by $(f;g) : A \to B \times C$ the function defined as $(f;g)(x) = (f(x), g(x))$ for every $x \in A$.

Theorem 3.4 *We have $H(f;g) = H(f|g) + H(g) = H(g|f) + H(f)$.*

Proof. Let us prove the first part of the equality. We have:

$$\begin{aligned} H(f;g) &= -\sum_{(b_i,c_j)\in B\times C}\frac{|A_j^i|}{|A|}\log\frac{|A_j^i|}{|A|} \\ &= -\sum_{j=1}^{n}\sum_{i=1}^{m}\frac{|A_j^i|}{|A|}\log\frac{|A_j^i|}{|A_j|}\frac{|A_j|}{|A|} \\ &= -\sum_{j=1}^{n}\sum_{i=1}^{m}\frac{|A_j^i|}{|A|}\log\frac{|A_j^i|}{|A_j|} \\ &\quad -\sum_{j=1}^{n}\sum_{i=1}^{m}\frac{|A_j^i|}{|A|}\log\frac{|A_j|}{|A|} \\ &= H(f|g) - \sum_{j=1}^{n}\log\frac{|A_j|}{|A|}\sum_{i=1}^{m}\frac{|A_j^i|}{|A|} \\ &= H(f|g) + H(g), \end{aligned}$$

where $A_j^i = \{x \in A | f(x) = b_i \wedge g(x) = c_j\}$, $A_j = \{x \in A | g(x) = c_j\}$. The proof of the equality $H(f;g) = H(g|f) + H(f)$ is analogous. ∎

Note that the above two theorems correspond to well known theorems about Shannon entropy of random variables, where $H(f;g)$ corresponds to the joint Shannon entropy of random variables.

Also, our definition of conditional entropy between two functions is a generalization of the notion of conditional entropy between a multiple-valued logic function and one of its variables, which was introduced in [2].

Let $A^{(1)}, \ldots, A^{(n)}$ and B be finite sets and $A = A^{(1)} \times \cdots \times A^{(n)}$. We denote by π_i the projection of the set A onto $A^{(i)}$. Let $f(x_1, \ldots, x_n) : A \to B$, where $x_i \in A^{(i)}$ be a multiple-valued logic function. The entropy of function f conditioned on one of its variables, say $x_i \in A^{(i)}$, is defined as $H(f|\pi_i)$. If $B = \{b_1, \ldots, b_m\}$, and $A^{(i)} = \{a_1, \ldots, a_n\}$ we have:

$$H(f|\pi_i) = -\sum_{j=1}^{n}\sum_{i=1}^{m} \frac{|A_j^i|}{|A|} \log \frac{|A_j^i|}{|A_j|},$$

where $A_j^i = \{x \in A | f(x) = b_i \wedge \pi_i(x) = a_j\}$, $A_j = \{x \in A | \pi_i(x) = a_j\}$, which corresponds to the definition given in [2].

4 Conclusion

Axiomatic definitions of entropy of functions between finite sets and of conditional entropies of such functions have been presented. In the axioms we used standard operations applied to discrete functions. We believe, that the paper will contribute to a better understanding of the notion of entropy of functions between finite sets. We also hope that our results will find applications in decomposition of MVL logic functions and in the estimation of power dissipation of digital circuits.

5 Acknowledgment

Szymon Jaroszewicz wishes to acknowledge the support received from the J. William Fulbright Scholarship Board as a visiting researcher at the University of Massachusetts at Boston (grant no. IIE#:15984114-ICB).

References

[1] Cheng K.T. and Agrawal V., *An Entropy Measure of the Complexity of Multi-Output Boolean Function*, Proc. of the 27th Design Automation Conference, pp. 302–305, 1990.

[2] Cheushev V., Shmerko V., Simovici D.A., and Yanushkevich S., *Functional Entropy and Decision Trees*, Proc. ISMVL'98, pp. 257–262.

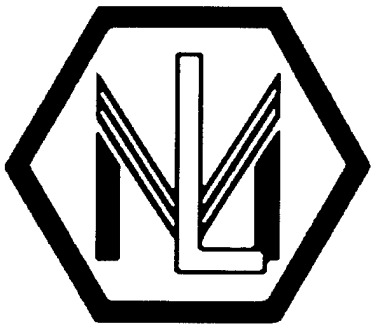
[3] Hwang C.H. and Wu A.C.H., *An Entropy Measure for Power Estimation of Boolean Function*, Proc. of Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 101–106, 1997.

[4] Lloris-Ruiz A., Gomez-Lopera J.F. and Roman-Roldan R., *Entropic Minimization of Multiple-Valued Logic Functions*, Proc. ISMVL'93, pp. 24–28.

[5] Mathai A.M. and Rathie P.N., *Basic Concepts in Information Theory and Statistics — Axiomatic Foundations and Applications*, Halsted Press, John Wiley & Sons, 1975.

[6] Simovici D.A. and Reischer C., *On Functional Entropy*, Proc. ISMVL'93, pp. 100–104.

SESSION IIB
CIRCUITS I
CHAIR: Yutaka Hata



Multiple-Valued Content-Addressable Memory Using Metal-Ferroelectric-Semiconductor FETs

Takahiro Hanyu, Hiromitsu Kimura and Michitaka Kameyama
Graduate School of Information Sciences
Tohoku University
Aoba-yama 05, Sendai 980-8579, Japan
E-mail : hanyu@kameyama.ecei.tohoku.ac.jp

Abstract

This paper presents a design of a non-volatile multiple-valued content-addressable memory (MVCAM) using metal-ferroelectric-semiconductor (MFS) FETs. An MFSFET is an important device with a non-destructive read scheme. Multiple-valued stored data are directly represented by remnant polarization states that correspond to threshold voltages of an MFSFET. Since one-digit comparison between multiple-valued input and stored data is performed by the combination of two different threshold operations, a one-digit comparator can be designed by two MFSFETs. The use of the one-digit comparator makes it possible to design a compact MVCAM cell. It is evaluated that the performance of the proposed MVCAM is superior to that of some binary and multiple-valued CAMs in terms of bit density, peripheral-circuit complexity, access speed, and functionality.

1. Introduction

Real-time programmable and non-volatile memories with a one-transistor cell structure are one of the key modules in the present high-performance system LSIs. Ferroelectric(FE) memory devices have increasingly attracted attention because they are non-volatile memories with the capability of high-speed read and write operations. For example, FE capacitors are about ten-times larger than that of the SiO_2 film, so that they have a potential advantage to implement high-density memories. A one-transistor memory cell can be implemented by only a single MFSFET with a non-destructive read operation.

It is well known that, in the present giga-scale system-on-a-chip integration era, accelerating the evolution in chip density causes a communication bottleneck between memory and logic modules. A logic-in-memory structure, in which storage functions are distributed over a logic-circuit plane, is a key technology to solve the above problem. CAM is a typical application of such a logic-in-memory VLSI architecture. Until now, several high-performance CAMs have been reported [1]-[6]. However, it has been difficult to solve the trade-off between real-time programmability and non-volatility with keeping a compact CAM cell circuit. A few challenging works, a MVCAM using FE devices, have been reported[6].

In this paper, a new design of an MVCAM using non-destructive FE devices such as MFSFETs is proposed. Multiple-valued stored data in the MVCAM cell are directly represented by threshold voltages of an MFSFET that can be programmed by controlling remnant polarization states. Multiple-valued one-digit comparison is performed by the logical product AND of two different threshold operations. Since the use of precharge-evaluate logic makes it possible to implement a two-input AND gate by just wiring, an MVCAM cell can be designed by two MFSFETs. An MVCAM word circuit is also designed by series connection of the cell circuits together with precharge-evaluate logic, so that an n-digit MVCAM word circuit can be realized by $2n$ MFSFETs. Moreover, the performance of the proposed MVCAM is compared with that of a binary dynamic CAM[2] and non-volatile MVCAMs[3],[5],[6] in terms of non-volatility, real-time programmability, a bit density, an execution speed, and peripheral-circuit simplicity. As a result, it is demonstrated that its performance is superior to that of the other CAMs.

2. Review of MFSFETs

2.1 Characteristics of MFSFETs

An MFSFET structure is similar to a conventional metal-oxide-semiconductor(MOS) FET with the exception that the gate insulating layer is replaced by an active ferroelectric thin film. Figure 1 shows a cross-sectional view and a symbol of an MFSFET. An FE

0-7695-0161-3/99 $10.00 © 1999 IEEE

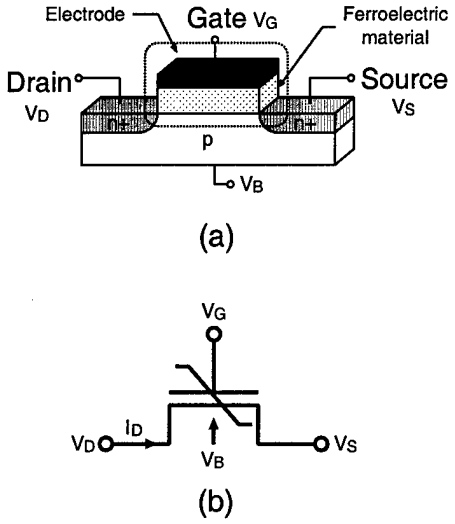


Figure 1. MFSFET structure, (a) Cross-sectional view, (b) Symbol.

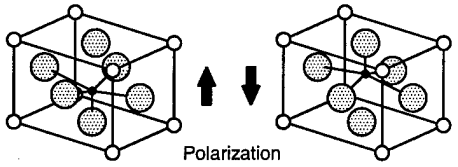


Figure 2. Perovskyte structure.

material consists of a Perovskyte structure as shown in Figure 2, which is transformed when an atom at the center of a lattice causes a polarization by a shift from an electrically neutral state according to the external electric field which is determined by an applied voltage and thickness of FE thin films. This characteristic is represented by a hysteresis loop. Figure 3 shows polarization vs voltage hysteresis loops of the FE film. One of the most important characteristics in the FE material is to have a remnant polarization within the saturated hysteresis loop when the voltage across the FE film is zero. It implies that the FE film can preserve information that corresponds to a remnant polarization state without voltage supply, i.e. an MFSFET is capable of a non-volatile memory device. Recently, MFSFETs have been reported using the variety of the FE material such as $Bi_4Ti_3O_12$[7], $LiNbO_3$[8], and $SrBi_2Ta_2O_9$[9].

Figure 3 also shows an example of a binary data assignment with remnant polarization states. As the remnant polarization is determined by an applied voltage across the FE film, whose magnitude is larger than the coercive voltage V_C, a logic state, "1" or "0", can be written into the MFSFET by the gate voltage V_G and the bulk voltage V_B. The required voltage applied to gate electrode of the MFSFET to write data is lower

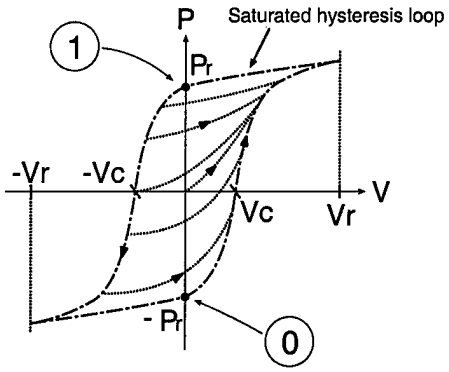


Figure 3. Hysteresis loop characteristics of an FE material.

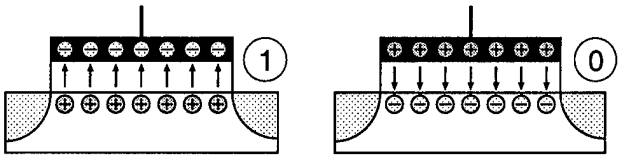


Figure 4. Compensation charge in the semiconductor surface.

than that of conventional non-volatile memory devices such as a floating-gate MOS transistor.

When the external voltage is removed, the remnant polarization will induce an electric field which attracts positive (or negative) compensation charge to the semiconductor surface as shown in Figure 4. Therefore, the carrier density of a semiconductor at the interface will be inverted, which makes it possible to shift the threshold voltage. Figure 5 shows the threshold voltage corresponding to the MFSFET state "0" and "1". In a read scheme, V_B is fixed to the ground level. When read gate voltage V_{RF} is applied to V_G, the switching state in the MFSFET depends on the polarization state in the FE film, i.e. stored data. Therefore, stored data in the MFSFET is read as "on" and "off" states. As V_{RF} is low enough, the polarizations state in the FE film remains the previous state. Therefore, the MFSFET can realize a non-destructive read operation.

Consequently, an MFSFET is capable of being used as a non-volatile memory device with a non-destructive read scheme. Principle of the operation is similar to a Floating-Gate MOS Transistor, but an MFSFET has the advantage of a high-speed and a low power write scheme due to the characteristic of FE films.

2.2 Multi-level threshold programming in MFSFETs

In an MFSFET, a stored value is represented by remnant polarization states. Therefore, several rem-

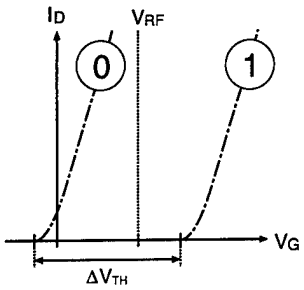


Figure 5. Relationship between threshold voltage and the remnant polarization states.

nant polarization states are required for multiple-valued data storage. For example, Figure 6 shows the relationship between four remnant polarization states and the assignment of four-valued data. Four remnant polarization states are obtained by the following method. The polarization in an FE film moves on branches as shown in Figure 4. This means that P_{r0} and $-P_{r0}$ are produced by the transition of the polarization through $a \rightarrow b \rightarrow c$, and $d \rightarrow e \rightarrow f$, respectively.

Consider the storage of "2" in the four-valued logic into MFSFETs. Figure 7 shows the timing diagram and polarization states for a write scheme. First, V_{r1} is applied to the gate electrode of the MFSFET, and then zero is applied. At that moment, the polarization stays on a. Second, $-V_{r0}$ is applied, and then zero is applied. The polarization changes from a to c through b. Consequently, the remnant polarization in FE films is P_{r0}, i.e. four-valued data "2" is stored.

Threshold voltages are shifted depending on remnant polarization states. If there are four remnant polarization states, we can utilize the four level threshold voltages. This characteristic is important to design a CAM cell structure using MFSFETs.

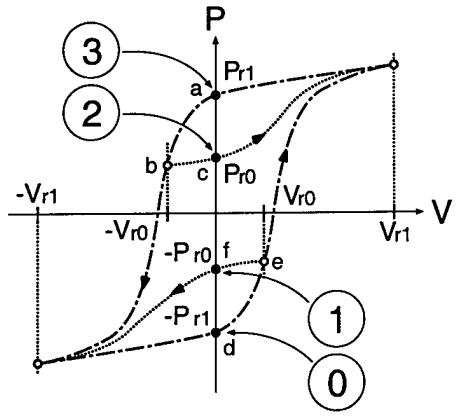


Figure 6. Four remnant polarizations states.

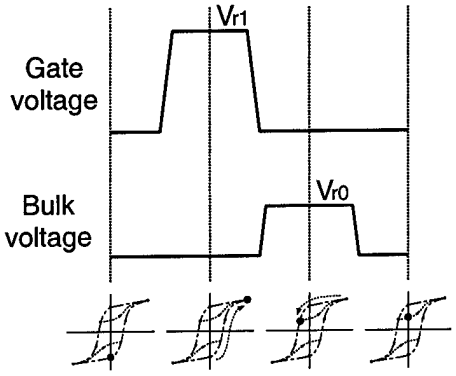


Figure 7. Timing diagram and the polarization state for a write scheme of a multiple-valued logic value.

3. Design of a Multiple-Valued Content-Addressable-Memory Using MFSFETs

CAM accomplishes a magnitude comparison between two kinds of R-valued input words, S(an n-digit external input word) and B_i(an ith n-digit stored input word), and generates a binary output word, Z(m-bit output) as its comparison result. Figure 8 shows a general structure of an MVCAM. In the case of R-valued encoding, an input word S and the ith stored word B_i are expressed as follows:

$$S = \sum_{j=1}^{n} R^{n-j} \cdot s_j, \quad (1)$$

and

$$B_i = \sum_{j=1}^{n} R^{n-j} \cdot b_{ij} \quad (2)$$

where s_j and $b_{ij} (1 \leq j \leq n)$ indicate the jth digits of S and B_i, respectively, and $s_j, b_{ij} \in \{0, 1, \cdots, R-1\}$. s_1 and b_{i1} are the most significant digits, and s_n and b_{in} are the least significant digits, respectively. The magnitude compassion between S and B_i is defined as

$$G(S, B_i) = \begin{cases} 1 & \text{if } S > B_i, \\ 0 & \text{otherwise.} \end{cases} \quad (3)$$

The ith binary value z_i in Z is equal to $G(S, B_i)$.

In the following subsection, we discuss about a hardware algorithm of the magnitude comparison, and a circuit design for a high-performance MVCAM using MFSFETs.

3.1 Hardware algorithm

The magnitude comparison $G(S, B_i)$ is represented by two kinds of threshold operations for each digit. One is the greater-than search operation, g_j, and the other

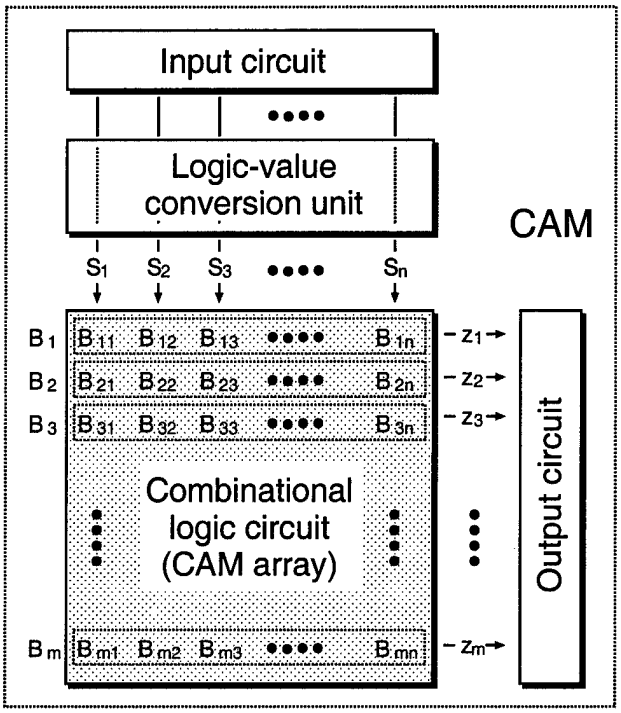


Figure 8. Structure of a CAM.

is the greater-than or equal-to search operation ge_j, between s_j and b_{ij}. These operations are defined as

$$g_j(s_j, b_{ij}) = \begin{cases} 1 & \text{if } s_j > b_{ij}, \\ 0 & \text{otherwise,} \end{cases} \quad (4)$$

and

$$ge_j(s_j, b_{ij}) = \begin{cases} 1 & \text{if } s_j \geq b_{ij}, \\ 0 & \text{otherwise.} \end{cases} \quad (5)$$

For example, let's consider the magnitude comparison between four-valued three-digit words, S and B, as

$$S = (s_1 s_2 s_3) = (321),$$
$$B = (b_1 b_2 b_3) = (320).$$

If B is greater than S, then one of the following conditions:

$$(a) \quad g_1 = 1,$$
$$(b) \quad ge_1 \wedge g_2 = 1,$$
$$\text{and} \quad (c) \quad ge_1 \wedge ge_2 \wedge g_3 = 1$$

must be at least satisfied. In this example, the above condition (c) is satisfied because of $(s_1 = b_1)$, $(s_2 = b_2)$ and $(s_3 < b_3)$.

In general, the magnitude comparison between n-digit words can be represented by using g_j and ge_j as

$$G(S, B_i) = g_1 \vee (ge_1 \wedge g_2) \vee (ge_1 \wedge ge_2 \wedge ge_3) \cdots \cdots \vee (ge_1 \wedge ge_2 \wedge \cdots \wedge ge_{n-1} \wedge g_n). \quad (6)$$

where symbols $\vee$ and $\wedge$ indicate binary logic operations, OR and AND, respectively. Eq.(6) is also transformed into

$$G(S, B_i) = g_1 \vee ge_1 \wedge (g_2 \vee ge_2 \wedge (\cdots \cdots (g_{n-1} \vee ge_{n-1} \wedge g_n) \cdots)). \quad (7)$$

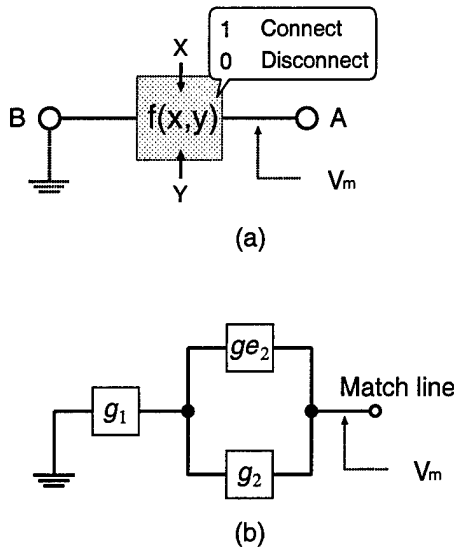


Figure 9. MFSFET-based pass gate, (a)Model, (b)Example.

We use a pass-transistor network[4],[5] to design the CAM circuit based on Eq.(7). In this concept, a function $f(x, y)$ which has two kinds of R-valued inputs x, y, and binary output are considered as a pass gate as shown in Figure 9(a). This pass gate is defined as

$$f(x, y) = \begin{cases} 1 & \text{A is connected to B,} \\ 0 & \text{otherwise.} \end{cases} \quad (8)$$

The result of $f(x, y)$ is obtained by the voltage of A which is precharged to V_m before operation. Where, B is connected to the ground level. The voltage of A depends on the result of $f(x, y)$. That is, the ML is discharged if $f(x, y)$ results in "1". Otherwise, the pass to discharge is broken and the A still remains V_m.

Two binary logic operators, AND and OR, can be easily realized by parallel and serial connections of pass gates. Figure 9(b) shows one example of realization of $g_1 \wedge ge_2 \vee g_2$ using pass-gates which correspond to g_1, ge_2, and g_2, respectively. Consequently, $G(S, B_i)$ in Eq.(7) is realized by combination of gates. Figure 10 shows a block diagram of the MVCAM word circuit, in which an MVCAM cell consists of pass gates which correspond to g_j and ge_j, respectively.

3.2 CAM cell circuit design using MFSFETs

Two kinds of pass gates, g_j and ge_j, can be designed using threshold operations[10] between the threshold

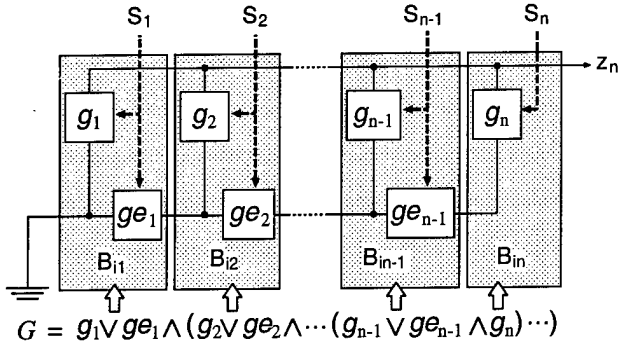


Figure 10. Block diagram of an MVCAM word circuit.

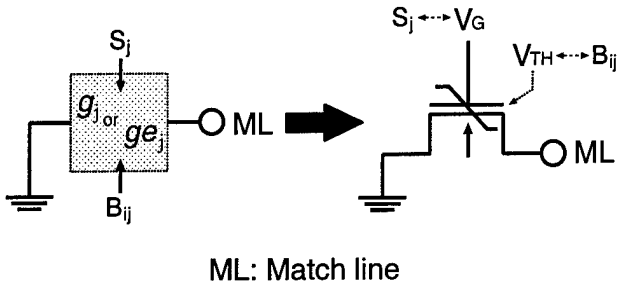


Figure 11. Realization of g_j and ge_j using MFSFETs

voltage V_{TH} and the gate voltage V_G of an MFSFET. The threshold voltage of the MFSFET represents the storage data. This means that the MFSFET can be considered to a pass gate which has two kinds of input V_G and V_{TH}, described in section 3.1. An external input s_i and a stored input b_{ij} correspond to V_G and V_{TH} as shown in Figure 11, respectively.

The function which is represented by the pass gate which consists of an MFSFET is determined by relationship between s_i and b_{ij} (that is, V_G and V_{TH}). Figure 12 shows relationship of the gate and threshold voltage which represents s_i and b_{ij} with regard of g_j and ge_j in the case of four-valued logic. Using this MFSFET, the CAM can be simply designed.

3.3 Overall structure of an MVCAM

Figure 13 shows the circuit diagram of a magnitude comparator between n-digit words. In this circuit, precharge-evaluate logic is employed to provide low power dissipation and high speed processing with less area overhead. When the clock ϕ is in a low state, the match line is precharged to V_m. When ϕ goes to a high state, the match line is discharged depending on the relationship between S and B_i. Data input for each cells are performed in parallel, and result of comparison are generated to z_i, simultaneously. Hence, this MVCAM accomplishes a read scheme in one step.

Figure 14 shows circuit diagrams for write opera-

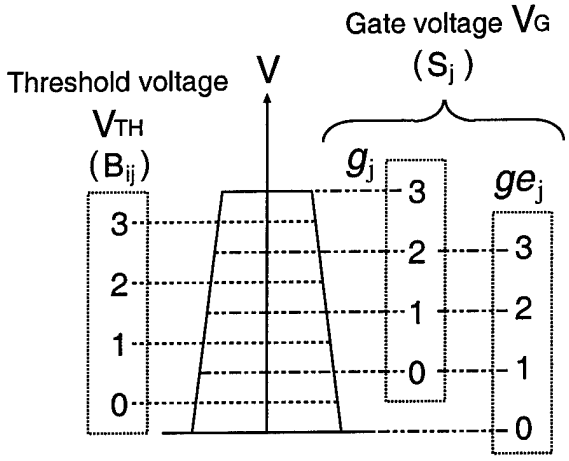


Figure 12. Four-valued threshold voltages and gate voltages of MFSFETs.

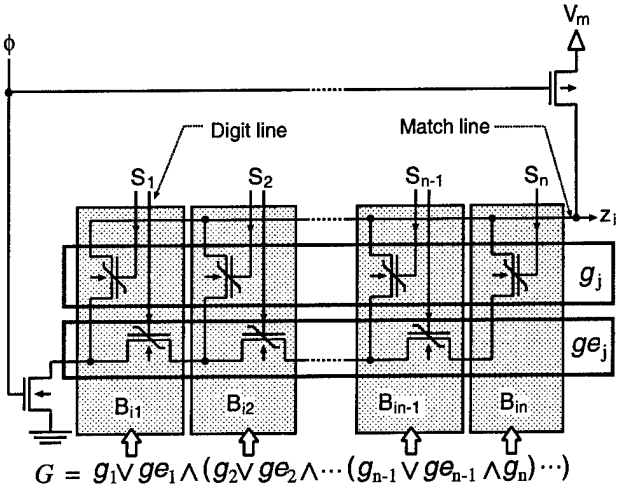


Figure 13. n-digit magnitude comparator based on a digit-parallel structure.

tions in the magnitude comparator. To discharge V_d and V_s for each MFSFETs, the voltage "3" is applied to MFSFETs which correspond to g_j. Then, the threshold voltage V_{TH} is programmed by V_g and V_b as shown in Figure 7. To avoid changing the threshold voltage of every non-selected MFSFETs, the digit line or the plate line which connects with those MFSFETs maintains floating.

Table 1 summarizes an estimated performance of CAMs. MFSFETs can realize real-time programmable and non-volatile memories with a non-destructive read scheme, so that the data does not require a reprogram scheme. Therefore, its peripheral circuit for a reprogram operation can be reduced. Moreover, since an MFSFET can perform a threshold operation between V_G and V_{TH} which correspond to s_j and b_{ij}, respectively, a CAM cell is designed by two MFSFETs

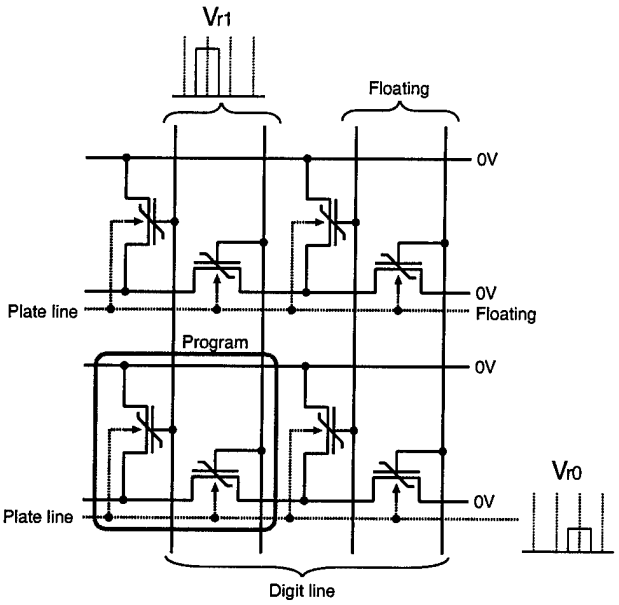


Figure 14. Write operation in a digit-parallel structure.

based on pass-transistor network. Consequently, the presented MVCAM can achieve the high-speed access scheme and high density .

4. Conclusion

A compact MVCAM with real-time programmability has been designed by using MFSFETs. Since a multiple-valued threshold operation is performed by using a single MFSFET whose multi-level threshold voltages correspond to remnant polarization states, a multiple-valued one-digit comparator can be designed by just two MFSFETs. Moreover, the use of the pre-charge-evaluate logic circuit makes an MVCAM word circuit compact.

As a future research target, it is also important to develop high-performance VLSI circuit with multiple-valued external inputs, multiple-valued stored inputs and binary outputs using MFSFETs.

References

[1] T.Hanyu, S.Aragaki and T.Higuchi, "Functionally separated, multiple-valued content-addressable memory and its applications," *IEE Proc.-Circuits Devices Syst.*, Vol.142, No.3, pp.165-172, Jun.1995.

[2] T.Yamagata, et al., "A 288-kb Fully Parallel Content Addressable Memory Using a Stacked-Capacitor Cell Structure," *IEEE J. Solid-State Circuits*, Vol. SC-27, No.12, pp.1927-1933, Dec.1992.

[3] T.Miwa et al., "A 1 Mb 2-Transistor/bit Non-Volatile CAM Based on Flash-Memory Technologies," *IEEE ISSCC Digest of Technical Papers*, pp.40-41, Feb.1996.

[4] T.Hanyu, N.Kanagawa and M.Kameyama, "Design of a One-Transistor-Cell Multiple-Valued CAM," *IEEE J.Solid-State Circuits*, Vol.31, No.11, pp.1669-1674, Nov.1996.

[5] T.Hanyu, K.Teranishi and M.Kameyama, "Multiple-Valued Floating-Gate-MOS Pass Logic and Its Application to Login-in-Memory VLSI", *Proc. IEEE International Symposium on Multiple-Valued Logic*, pp.270-275, 1998.

[6] A.Sheikholeslami, P.G.Gulak and T.Hanyu, "A Multiple-Valued Ferroelectric Content-Addressable Memory", *Proc. IEEE International Symposium on Multiple-Valued Logic*, pp.74-79, 1996.

[7] S.Y.Wu, "A New Ferroelectric memory Device, Metal-Ferroelectric-Semiconductor Transistor," *IEEE Trans. Electron Devices*, Vol.21, No.8, pp.449-505, Aug. 1974.

[8] K.H.Kim, "Metal - Ferroelectric - Semiconductor (MFS) FET's Using $LiNbO_3$/Si(100) Structures for Nonvolatile Memory Application,", *IEEE Electron Dev.Let.*, Vol.19, No.6, pp.204-206, Jun. 1998.

[9] G.Fuji, T.Shimamura, E.Tokumitsu, and H.Ishikawa, "Fabrication and characterization of $SrBi_2Ta_2O_9$/Si MFSFETs," *Extended Abstracts of 44th Spring Meeting of the Jpn.Soc.Appl.Phys. and Related Societies*, p.488, Mar. 1997.

[10] T.Higuchi and M.Kameyama, *Multiple-Valued Digital Processing System,* Shokodo Co. Ltd., Tokyo, 1989(in Japanese).

Table 1. Comparison of CAMs.

CAM	Type	Non-volatile	Real-time Programmability	Bit-density	Speed/bit	Peripheral Circuit Simplicity
Dynamic [2]	Binary	No	Yes	+	+	+
Floating-gate MOS Tr. based [3]	Binary	Yes	No	++	+	+
Floating-gate MOS Tr. based [5]	Multiple-valued	Yes	No	+++	+++	++
FE capacitor based [6]	Multiple-valued	Yes	Yes	+++	++	+
MFSFET based Digit-parallel	Multiple-valued	Yes	Yes	+++	+++	++

"New lamps for old!"
(Generalized Multiple–valued Neurons)

Claudio Moraga and Ralph Heider
Department of Computer Science and Computer Engineering
University of Dortmund
44221 Dortmund
Germany

{moraga | heider}@LS1.informatik.uni-dortmund.de

Abstract

Contributions to multiple–valued threshold logic in the seventies are reviewed at the light of developments in the area of artificial neural networks. It is shown that it is possible to adapt methods of design of feedforward neural networks to generate networks of multiple-valued neurons to realize any multiple-valued function.

1. Introduction

The rediscovery of (and the publicity given to) the gradient descend algorithm to tune networks of especial non–linear devices called *"neurons"* to meet specifications of a problem based on representative examples of performance made in the middle eighties by D.E. Rumelhart and his colleagues [RHW 86] (see however [BrH 69], [Par 85], [Wer 74]), has lately motivated the MVL–community to retake the study of multiple–valued neurons (see *e.g.* [ABJ 93], [JBA 93], [TIT 95], [Obr 96], [NRS 98], [MaC 98]) what in the sixties and seventies was simply known as multiple–valued threshold logic (see *e.g.* [Han 63], [Mer 64], [AiA 70], [Kit 70], [SaA 70], [Mor 72], [Mor 75], [Mor 77], [Mor 79c]). The renewed interest in this area has however a different focus, that opens new research questions and possibilities. It is not –(at least not yet)– related to a specific hardware realization and this indeed has allowed facing questions that in the former decades were very soon discarded. On the other hand the present state of affairs has not yet reached the level of generality considered by the neural networks community. It seems to be the right time to point out some common aspects of interest that are relevant to both the MVL and the NN–communities, particularly the question of designing networks of (possibly generalized) multiple–valued neurons. The rest of the paper is organized as follows: in the next section the required basics will be formalized. Section 3 is devoted to cover the relevant aspects of the design of feedforward neural networks. Representative examples will be discussed in section 4. The paper will be closed with a summary of conclusions.

2. Formal background

Definition 1: Artificial Neural Network

An artificial neural network is a computational model inspired in structures existing in the brains of mammals. It consists of memoryless elementary processors called (by analogy) *"neurons"* and links interconnecting them. Links are weighted and induce a signal scaling. Algorithms are known to tune the weights in order to minimize the square error of performance.

Definition 2: Neuron

A neuron (of an artificial neural network) is an n–inputs, 1–output elementary processing unit that represents the composition of three functions:

- an input function $I: C^n \rightarrow C$
- an activation function $A: C \rightarrow C$
- an output function $O: C \rightarrow C$,

where C denotes a compact subset of the reals.

In most neural networks I is taken to be an affine transformation and it computes the weighted sum of the input variables with possibly a bias. Usually A is chosen to be monotone. In that case it may be hard symmetric or asymmetric (*sign* or *Heaviside* function, respectively) otherwise soft symmetric or asymmetric (*htan* or *sigmoid*, respectively). The output function O is either the identity, a scaling linear function or a limiting non–linear one.

Definition 2.1: Multiple–valued neuron

A multiple–valued neuron is an n–input, 1–output device that may be specified as the composition of the three following functions:

- I: $(Z_p)^n \rightarrow R$, for a predefined integer $p > 1$
- A: $R \rightarrow Z_p$
- O: $Z_p \rightarrow Z_p$

0-7695-0161-3/99 $10.00 © 1999 IEEE

I is an affine function computing the weighted sum of the inputs, where the weights are (in principle) real values; but without loss of generality may be restricted to be integers [Mor 77]. *A* is a staircasewise thresholding function and *O* is a one–place function in Z_p.

It becomes apparent that if *O* is the identity function, Definition 2.1. specifies a p–valued threshold function. Multiple–valued threshold functions have been very thoroughly studied for p=3 (see *e.g.* [AiA 70], [NaM 74], [Mor 77]). These functions, as in the case of binary threshold functions are characterized by their Chow–Parameters [AiA 70], or equivalently, by the first order Chrestenson spectral coefficients of the function [Mor 79a], [Mor 79b]. Results for p>3 may be found in [Mor 79c], [MSO 82] and [Mor 89]. It has been shown that if *O* is allowed to be the identity or a mirroring negation, there are 471 2–place and 85,629 3–place ternary threshold functions [AiA 70], [Mor 77], and 18,184 2–place quaternary threshold functions [MSO 82], respectively. Of course, threshold functions constitute a functionally complete set, however as in the case of Rosenblatt's *perceptron* [Ros 57], [Ros 58] the fact that threshold functions are defined in terms of hard steps impaired the use of a gradient descend algorithm to systematically develop efficient networks based on threshold gates. This was, among others, one of the main reasons to start adding degrees of freedom to the definition of threshold functions to increase their range of realization possibilities. Probably one of the first contributions in this direction was the introduction of polynomial separability [Mor 72], [Mor 75], [Mor 77], which corresponds to the idea of higher order neurons (in the world of artificial neural networks) [RöM 91]. In this case *I* is no longer affine, but a quadratic or higher order function. However due to the discrete structure of Z_p^n it may be realized with the help of unary functions, since the separating polynomials may be properly replaced by "Manhattan"–trayectories reaching an equivalent separation [Mor 77], [Mor 79c]. The other early contribution was the introduction of multilineal separability [NaM 75], [Mor 77], [Mor 79c], [Mor 89], which actually leads to realization of multiple–valued functions as a network of elementary two-valued threshold functions connected to a (quantizing adding gate realized as a) p–valued threshold gate, where every threshold selects the corresponding value from Z_p. (Of course threshold functions are trivially multilineal separable). It becomes apparent that this may be extended to combine polynomial and multilineal separability in one network to increase flexibility. This was done and called "Adaptive Separability" in [Mor 79c]. Related aspects have recently been addressed [AbA 98] within the scope of Frontier Algorithms [McW 92].

To conclude this section another view of classical p–valued threshold functions and of the above presented extensions will be given from a more formal point of view.

Definition 3: Subdomains

$\forall k \in Z_p$ let $\Psi_k = \{x \in Z_p^n \mid f(x) = k\}$

It becomes apparent that a subdomain may be empty if the function is not suprayective.

Definition 4: p–valued linear separable or threshold functions

A p–valued function f: $Z_p^n \rightarrow Z_p$ is linear separable or threshold iff there exists a set of parallel hyperplanes separating the subdomains of Z_p^n in a monotonic way. (See example in figure 1.)

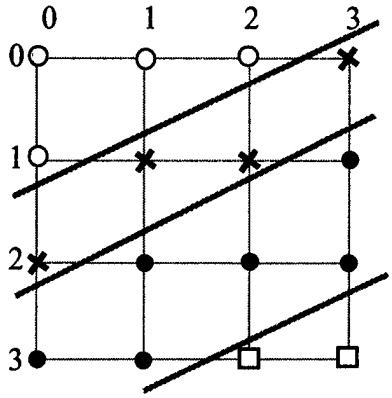


Figure 1: A linear separable or threshold 4–valued function (o = 0, x = 1, ● = 2, □ = 3)

Definition 4.1: p–valued polynomial separable functions

A p–valued function f: $Z_p^n \rightarrow Z_p$ is polynomial separable if there exists a set of parallel hyper*surfaces* separating the subdomains of Z_p^n in a monotonic way. (See example in figure 2.)

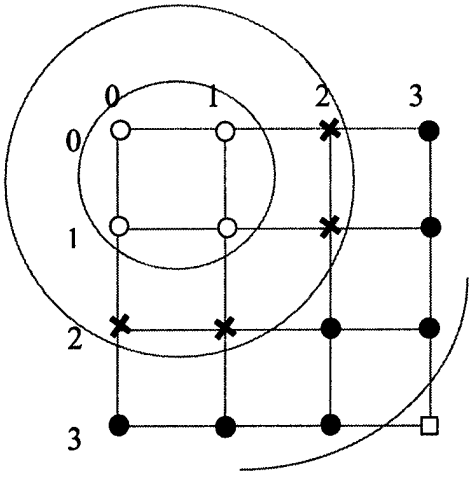


Figure 2: Polynomial separation of a 4–valued function (o = 0, x = 1, ● = 2, □ = 3)

Definition 4.2: p–valued multilineal separable functions

A p–valued function f: $Z_p^n \rightarrow Z_p$ is multilineal separable if there exists a set of *non–necessarily* parallel hyperplanes separating the subdomains of Z_p^n. Notice that in this case more than one hyperplane may be used to separate two neighbour subdomains. (See example in figure 3.)

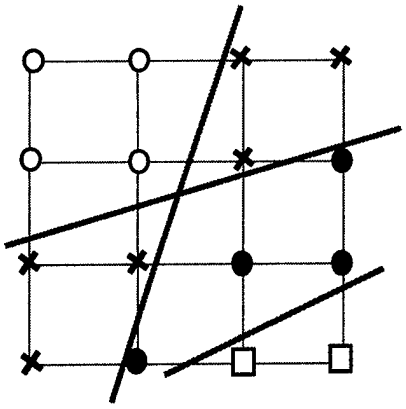


Figure 3: Multilineal separation of a 4–valued function
(O = 0, x = 1, ● = 2, □ = 3)

The combination of the above extension leads to Adaptive Separable functions. Figure 4 illustrates the adaptive separation of the same 4–valued function of figure 2, where two polynomials are replaced by literals (which are 1–place threshold functions) and the third by a line. The following realization is obtained:

if ${}^0x^1_1 - x_2 \leq 0.5$ then $f_{11}(x) =$ 1 else 0

if ${}^0x^2_1 - x_2 \leq 1.5$ then $f_{12}(x) =$ 1 else 0

if $x_1 + x_2 \geq 5.5$ then $f_2(x) =$ 1 else 0

where ${}^ix^j = 3$ iff $i \leq x \leq j$ else 0. This leads to:

$$f(x) = f_{11}(x) + f_{12}(x) + f_2(x) \qquad (1)$$

Notice that if the complement of x, written x^ is defined to be 3-x, then the former equations for $f_{11}(x)$ and $f_{12}(x)$ may be given another expression, where the auxiliary unary function will be monotone increasing.

Since ${}^0x^1_1 = 3 - ({}^0x^1_1)^\wedge = 3 - {}^2x^3_1$ the inequality

${}^0x^1_1 - x_2 \leq 0.5$

turns into

$3 - {}^2x^3_1 - x_2 \leq 0.5$

leading to

if ${}^2x^3_1 + x_2 \geq 2.5$ then $f_{11}(x) =$ 1 else 0

and similarly

if ${}^3x^3_1 + x_2 \geq 1.5$ then $f_{12}(x) =$ 1 else 0

3. Design of neural networks

In the general case neural networks are designed to solve continuous non–linear separable problems specified as f_c: $C^n \rightarrow C$. Neural networks have the structure of a graph, where each node is assigned the functionallity of a neuron. The most

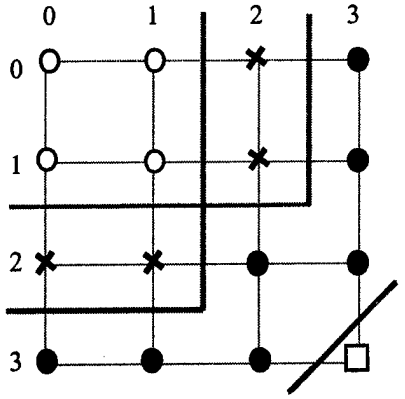


Figure 4: Adaptive separation of the function of figure 2
(O= 0, x = 1, ●= 2, □= 3)

graph structure, is known as "feedforward" network and is adjusted –("trained")– by means of a finite set of representative input–output examples that drive a gradient descend algorithm searching for a minimum of the square error (produced by the unadjusted network). There are several crucial problems that accompany this process. The number of required i/o–examples to properly adjust the network is proportional to a quantity known as the Vapnik–Chervonenkis Dimension [Blu 89] and this depends on structural properties of the network and of the activation function of the neurons [KoS 97]. A network is said to be properly adjusted if not only is able to reproduce the training examples, but give reasonable output to unknown inputs drawn from the same set of cases from where the examples were taken. This is known as *PAC Learning* [Val 84] in the algorithmic learning theory. Finally, in order to apply the gradient descend algorithm, it is needed that all activation functions are overall differentiable. (This immediately rules out all hard limiters like the staircase function, which is typical of multiple–valued neurons.)

Neural networks designed to solve the especial class of discrete non–linear problems specified as f_d: $Z^n \rightarrow Z$ are less critical in the sense that they do not have to generalize in the closest dense sorrounding of the training examples, but the other problems mentioned above basically remain.

Neural networks for multiple–valued functions are specified as f_m: $Z_p^n \rightarrow Z_p$ and the main difference with the former case (beyond the finitness of domain and range) is that usually *all* p^n evaluations of f_m are known and available as training examples (although not all of them may be necessary as training examples (see *e.g.* [AbA 98]).) Thus, the generalization problem disappears. The problem that the

activation functions cannot be hard limiters is more a "model" problem than a "realization" problem. (Recall the strong efforts that are needed to "correct" the step response of an amplifier from being sigmoid like to become close to a "corner".) Since without loss of generality all weights may be chosen to be integers [Mor 77], it is important to construct soft staircases satisfying two conditions:

i) It should be possible to have steps to detect consecutive integer values (of an input weighted summation), and

ii) The transitions of a step from low to high (or vice versa) measured as the inverse value of the tangent at the inflection point, should be fast enough not to interfer with threshold detection. This may be accomplished by taking a transition margin no larger than 0.333.

4. Analysis of examples

Unless an evolutionary design method is used [Kit 91], [Yao 93], [HeM 96], [Hei 97], [FrM 97], in the case of feedforward neural networks a tentative network must be *a priori* selected and then adjusted. A differentiable soft staircase activation function may be constructed based on the following sigmoid:

$$S(y) = [1 + e^{-20y}]^{-1} \qquad \text{from where } S(0) = 0.5$$

It is simple to show that $S'(y) = 20S(y)(1-S(y))$.
It follows that:

$$S'(0) = 20\ 0.5\ (1-0.5) = 5$$
$$1/S'(0) = 0.2 < 0.333$$

Moreover $S(1) = [1 + e^{-20}]^{-1} = [1 + 2.06\ 10^{-9}]^{-1}$
$S(1) = 0.99999999794$ (*i.e.* ≈ 1)

A soft staircase activation function for multiple-valued neural networks may be constructed as:

$$A(y) = \sum_{i=1}^{p-1} S(y - b_i) \qquad (2)$$

where b_i is a bias to properly displace the elementary sigmoids. This is illustrated in figure 5 for p=4, $b_1 = 0.5$, $b_2 = 1.5$ and $b_3 = 2.5$

The design of a quaternary full adder has been chosen as a first test problem. The architectures shown in figure 6 will be used as basic neural netrworks and in both cases, weights and bias values must be learned to satisfy the truth table of the full adder. It is known [Mor 89] that the network in figure 6a is minimal, meanwhile the network in 6b has a redundant node. All neurons have an affine input function, a soft staircase activation function and a linear output function. The training was done with the Stuttgart Neural Network Simulator SNNS [Zel 95] with Rprop [Rie 94] as gradient descend algorithm. The obtained weight and bias values are summarized in table 1. (Since SNNS is a design and simulation tool for classical neural networks, i.e. the activation functions are soft steps that have only two asymptotic values, the quaternary neurons were simulated by three two–valued neurons followed by an adder, as suggested by eq. (2) or simply by using the property of multilineal separability (see eq. 1). Moreover the inherent symmetry of the problem was taken in consideration to speed–up the training.).

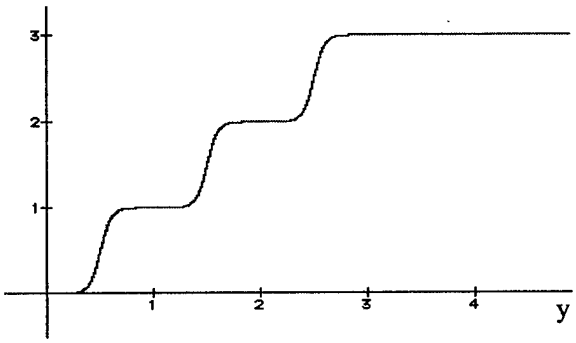


Figure 5: Soft staircase for 4–valued neurons

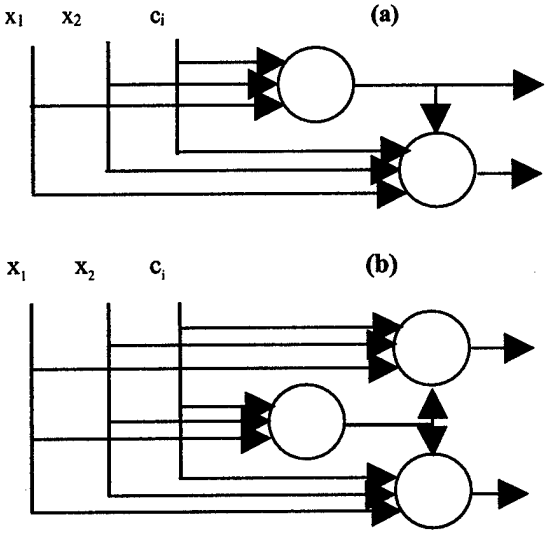


Figure 6: Full adder quaternary networks
(See Table 1 for weights and bias values)

Table 1: Summary of learned weights and biases

Circuit	hidden	carry out	sum
	weights (bias)	weights (bias)	weights (bias)
6a	---	1 1 1 (3.5; 6.5; 6.5)	1 1 1 -4 (0.5; 1.5; 2.5)
6b	1 1 1 (3.5; 6.5; 6.5)	0 0 0 1 (0.5; 1.5; 2.5)	1 1 1 -4 (0.5; 1.5; 2.5)

The second test example is the function shown in figure 2 with the separation suggested in figure 4 and discussed earlier in the text. The first direct circuit realization leads to the network shown in figure 7a. It has been shown [Mor 77], that this may be reduced to the network of figure 7b.

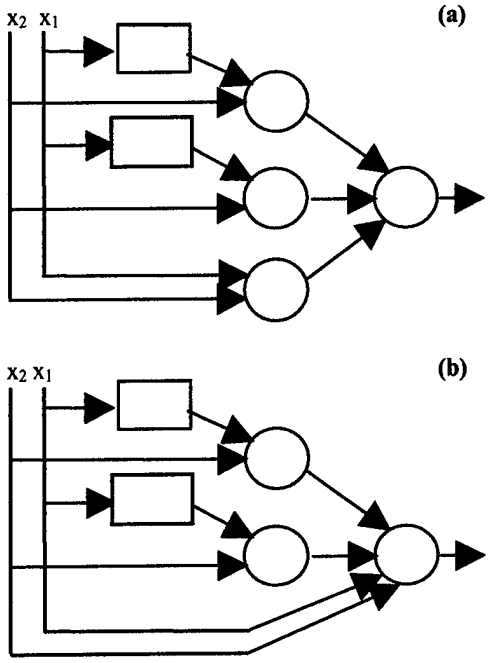


Figure 7. (a) Adaptive separation of test function
(b) Reduced adaptive realization.

All the weights of the threshold functions are 1. The bias values for the "hidden" threshold gates match the values given in the equations leading to or following eq. 1, respectively. The bias values of the output gate of the reduced network are: 2.5; 4.5 and 7.5.

5. Conclusions

Studies in multiple–valued threshold logic done in the seventies find a new focus of attention from the point of view of neural networks. Making use of formal developments in this area, it is possible to design multiple–valued neural networks of different classes in an algorithmic way (gradient descend). The idea of considering an output function for neurons, which may produce any permutation –(and not only the negation)– of values, suggested by the MV–community [AbA 98], [NRS 98], may be of interest for possible applications in discrete neural networks with finite range. (Classification problems are possibly the simplest but most frequent problems where this variation might find a reasonable use.) It should be recalled that designing multiple–valued circuits as feedforward (discrete) neural networks by means of a gradient descend algorithm requires an *a priori* decision on the configuration of the circuit (which may *a posteriori* be pruned if redundant gates are detected). If a really free design of a multiple–valued circuit as a feedforward (discrete) neural network is looked for, then an evolutionary strategy should be used [WaM 96], [HeM 96], [MoW 98].In fact, by using the evolutionary design method developed in [Hei 97] the minimal solution for the quaternary full adder was obtained *without* giving a priori the structure of the network.

References

[ABJ 93] Abd-El-Barr M., Bolton R., Jain A.: Current-mode CMOS realization of a MVL Neurode. *Proc. IEEE Wesca* , 201-207, 1993

[AbA 98] Abd–El–Barr M.I., Abd–El–Barr M.M.: A Frontier Algorithm for Optimization of Multiple–valued Logic Functions. *Proc. 28th. Int. Symp. on Multiple–valued Logic*, 245–249, Fukuoka, Japan. IEEE–CS–Press, 1998

[AiA 70] Aibara T., Akagi M.: Generation of ternary threshold functions up to three variables. *Jr. IECE Japan,* **53-C**, 591–598, 1970

[Blu 89] Blumer A., Ehrenfeucht A., Haussler D., Warmuth M.: Learnability and the Vapnik–Chervonenkis dimension. *Jr. of the ACM* **36**, 929–965, 1989

[BrH 69] Bryson A.E., Ho Y.C.: *Applied Optimal Control.* Blaisdell, N.Y., 1969

[FrM 97] Friedrich Ch., Moraga C.: Using Genetic Engineering to find modular structures and activation functions for architectures of artificial neural networks. *LNCS* **1226**, 150–161. Springer Verlag, Berlin, 1997

[JBA 93] Jain A., Bolton R., Abd-El-Barr M.: On multiple-valued logic design of Neural Networks. *Proc. 36th Midwest Symp. on Circuits & Systems*, 1489-1492, Detroit MI., 1993

[Han 63] Hanson W.H.: Ternary threshold logic. *IEEE Trans. Elec. Computers* **EC-12**, (6), 191–197, 1963

[Hei 97] Heider R.: *Evolutionäre Synthese neuronaler Netze unter Verwendung von Graph–Grammatiken.* Shaker Verlag, Aachen, 1997

[HeM 96] Heider R., Moraga C.: Evolutionary Synthesis of Neural Networks based on Graph Grammars. *Proc. Int. Panel Conference on Soft and Intelligent Computing,* 119–126. Budapest, ISBN 963 420 510 0, 1996

[Kit 70] Kitahashi T.: Monotonicity of ternary logic functions and its applications to analysis of ternary threshold functions. *Systems–Computers–Control* **1**, 57–63, 1970

[Kit 90] Kitano H.: Designing Neural Networks using a Genetic Algorithm with Graph Generation System. *Complex Systems* **4**; 461–476, 1990

[KoS 97] Koiran P., Sontag E.D.: Neural Networks with quadratic VC Dimensions. *Jr. of Computer Systems and Science* **54**, 190–198, 1997

[MaC 98] Macchiarulo L., Civera P.: Ternary decision diagrams with Inverted Edges and Cofactoring - An application to Discrete Neural Networks Synthesis. *Proc. 28th. Int. Symp. on Multiple–valued Logic,* 58–63, Fukuoka, Japan. IEEE–CS–Press, 1998

[McW 92] McCrosky C., Wang Y.: Finding frontiers without searching. Technical Report, University of Saskatchewan, Saskatoon, 1992

[Mor 72] Moraga C.: Non-linear Ternary Threshold Logic. *Conf. Record 1972 Symposium on the Theory and Appl. of Multiple-valued Logic Design.* Buffalo, N.Y., USA, 1972

[Mor 75] Moraga C.: Polynomial Separability of Ternary Functions. *LNCS* **34**, 523-533, Springer, Berlin, 1975

[Mor 77] Moraga C.: A Monograph on Ternary Threshold Logic. In *"Computer Science and Multiple-Valued Logic".* (D.C. Rine, Ed.), 355-394. North-Holland, Amsterdam, 1977 and 1984

[Mor 79a] Moraga C.: Spectral Characterization of Ternary Threshold Functions. *Electronics Letters* **15**, (22), 712-713, 1979

[Mor 79b] Moraga C.: Characterization of Ternary Threshold Functions by using a Partial Spectrum. *Electronics Letters* **15**, (24), 803-805, 1979

[Mor79c] Moraga C.: Extensions on Multiple-valued Threshold Logic. *Proc. 9th. Int. Symp. on Multiple-valued Logic*, 232-240. Bath, England. IEEE-CS-Press, 1979

[Mor 89] Moraga C.: Multiple-valued Threshold Logic. In *"Optical Computing. Digital and Symbolic"* (R. Arrathoon, Ed.), 161-183. Marcel Dekker Inc., New York, 1989

[MoW 98] Moraga C., Wang W.: Evolutionary Methods in the Design of Quaternary Digital Circuits. *Proc. 28th. Int. Symp. on Multiple-valued Logic*, 89-94. Fukuoka, Japan. IEEE-CS-Press, 1998.

[MSO 82] Moraga C., Schulte-Ontrop R.: On some cardinality questions in multiple-valued extended threshold logic. *Proc. 12th. Int. Symp. on Multiple-valued Logic*, 523-529. Paris, France. IEEE-CS-Press, 1982

[NaM 74] Nazarala J., Moraga C.: Minimal Realization of Ternary Threshold Functions. *Proc. 4th. Int. Symp. on Multiple-valued Logic,* 347-358, Morgantown, USA, 1974

[NaM 75] Nazarala J., Moraga C.: Bilineal Separability of Ternary Functions. *Proc. 5th. Int. Symp. on Multiple-valued Logic,* 305-315, Bloomington IN, USA, 1975

[NRS 98] Ngom A., Reischer C., Simovici D.A., Stojmenovic I.: Learning with permutably homogeneous multiple–valued multiple–threshold perceptrons. *Proc. 28th Int. Symp. on Multiple–valued Logic,* 161–166, Fukuoka, Japan. IEEE–CS–Press, 1998

[Obr 96] Obradevic Z.: Computing with nonmonotone multivalued neurons. *Multiple–valued Logic - An International Journal* **1**, (4), 271–284, 1996

[Par 85] Parker D.B.: Learning Logic. Technical Report TR–47, Center for Computational Research in Economics and Management Science. MIT, Cambridge MA, 1985

[RöM 91] Röckmann D., Moraga C.: Using Quadratic Perceptrons to reduce Interconnection Density in Multilayer Neural Networks. *LNCS* **540**, 86-92, (1991)

[Ros 57] Rosenblatt F.: The Perceptron: a perceiving and recognizing automation (Project PARA). Cornell Aeronautical Laboratory Report, 85–460–1, 1957

[Ros 58] Rosenblatt F.: The Perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review* **65**, 386–408, 1958

[RHW 86] Rumelhart D.E., Hinton G.E., Williams R.J.: Learning internal representations by error propagation. In *Parallel Distributed Processing* (D.E. Rumelhart, J.L. McLelland, Eds.), MIT Press, Cambridge, 1986

[Rie 94] Riedmiller M.: Rprop - Description and implementation details. Technical Report, Institut für Logistik, Komplexität und Deduktionssysteme, University of Karlsruhe, 1994

[SaA 70] Santos J., Arango H.: A graphic method for the synthesis of ternary threshold functions. *IEEE Trans. Comp.* **C-19**, (10), 975–976, 1970

[TIT 95] Tang Z., Ishizuka O., Tanno K.: Learning multiple-valued logic networks based on backpropagation. *Proc. 25th. Int. Symp. on Multiple–valued Logic,* 270–275. Bloomington IN, USA. IEEE–CS–Press, 1995

[Val 84] Valiant L.G.: A theory of the learnable, *Communications of the ACM* **27**, 1134–1142, 1984

[WaM 96] Wang W., Moraga C.: Design of Multiplevalued Circuits using Genetic Algorithms. *Proc. 26th. Int. Symp. on Multiple-valued Logic*, 216-221. Santiago de Compostela, Spain. IEEE-CS-Press, 1996

[Wer 74] Werbos P.J.: Beyond Regression: New tools for prediction and analysis in the behavioural sciences. Dissertation, Harvard University, 1974

[Yao 93] Yao X.: A Review of Evolutionary Artificial Neural Networks. *International Journal of Intelligent Systems* **8**, (4), 539–567, 1993

[Zel 95] Zell A. *et al*: SNNS User Manual Version 4.0. Technical Report 6/95, IPVR, University of Stuttgart, 1995

Supplementary Symmetrical Logic Circuit Structure

Edgar "Dan" Olson, Inventor
EDO LLC, 15309 Bittner Place
Moorpark, CA 93021-3209 U.S.A.
E-mail: enigmatist@juno.com

Abstract

The Supplementary Symmetrical Logic Circuit Structure (SUS-LOC) is a fully active, self sustaining architecture intended primarily for the design and fabrication of logic synthesizing circuits with a radix greater than two. Any 'r'-valued logic function of 'n'-places (where: 'r' is the radix and an integer greater than 1, and 'n' is an integer greater than 0) can be implemented with the SUS-LOC structure.

1.0: Introduction

The features "self sustaining" and "fully active" are essential to the structure used to implement Multiple Valued Logic (MVL) circuits. For the purpose of this paper these terms are defined as follows:

A self sustaining architecture is one capable of implementing at least one functionally complete set of connectives.

A fully active architecture is one that ideally achieves a quiescent power consumption of zero; realistically, the lowest quiescent power consumption is that of leakage currents only.

To date, the only self sustaining, and fully active circuit structure has been the Complementary Symmetrical Metallic Oxide Semiconductor circuit structure (CMOS) when used to implement binary logic functions. However, CMOS is not capable of implementing MVL functions and remaining fully active. The only self sustaining circuit structure intended for the implementation of MVL functions has been Current Mode CMOS Logic (CMCL). However, CMCL is not fully active. See references [3], [4], and [5].

MVL circuits must be high speed with a low power requirement, have well defined logic levels and be easily manufactured to be competitive with binary logic. To achieve MVL circuits with such attributes, MVL circuits must be based upon a fully active, self sustaining architecture.

2.0: The SUS-LOC Structure

The Supplementary Symmetrical Logic Circuit Structure (SUS-LOC) uses a technique called "supplementation" to detect and generate stable, well defined intermediate logic levels. The essence of supplementation is the use of two switches, connected to conduct in series, for each intermediate logic level between two terminus logic levels.

The simplest form of a SUS-LOC circuit with one intermediate logic level consists of: 1) one switch for each of two terminus logic levels; and 2) two switches connected to conduct in series, and sharing the control signal with the termini, for the intermediate logic level thereby supplementing the termini logic levels.

The advantages of the SUS-LOC structure include:

1. a quiescent power requirement consisting of leakage currents only;
2. improved fan-out characteristics;
3. well defined logic level domains;
4. switching times in the low nano-seconds;
5. multiple implementations of most functions;
6. redundant, functionally complete sets of logic; and
7. all one place functions (OPF) of radix 'r' are implemented without a multiple place function (MPF).

2.1: Branch Definitions

Any element that connects or disconnects an output terminal to or from a source of power, in response to an input stimulus, is a branch. The SUS-LOC structure contains two primary branches named Terminus and Intermediate, and one secondary branch named Composite. Generally, all functions require two terminus branches per input term, (the exceptions are not within the scope of this paper).

A terminus branch consists of one switch per input term, which connects an output terminal to a source of power representing a logic level and is responsive to either one or a group of contiguous input logic levels.

0-7695-0161-3/99 $10.00 © 1999 IEEE

The presence and numeration of intermediate branches is dependant upon the radix and specific logic function being synthesized. An intermediate branch consists of two switches, per input term, connected in series, which connect an output terminal to a logic level voltage between those conducted by the terminus branches. An intermediate branch is responsive to either one or a group of contiguous input logic levels.

Composite branches are a combination of primary branches connected in series, parallel, or series-parallel as required by the multiple place function in which they occur.

2.2: Switches

The switches selected for this disclosure of the SUS-LOC structure, due to their low cost, high reliability, and ease of manufacture are Insulated Gate Field Effect Transistors, (FET(s)). However, any switch with similar properties is a suitable substitute.

The channel type, mode, and threshold voltage, ($V_{GS(TH)}$), of each FET is fabricated, or selected, as required by the circuitry being fabricated.

Because FETs have been selected, a method of protecting the inputs against the damaging effects of input over/under voltage and electro-static discharge should be employed.

2.3: Symbols

Figures 1a through 1d show the symbols used to represent the various FETs in schematics. Near the gate of each FET is a "+V" or "-V" label to indicate the polarity, and magnitude of the FET's $V_{GS(TH)}$.

2.4: Mode and Channel Type Determination

The relationship of the input logic levels (I) responded to verses the output logic level (O) of a branch determines the channel type 'P' or 'N' and mode, Enhancement or Depletion (E or D), of the FET(s) used to form the branch.

There are typically two possible I/O relationships for a terminus branch, when: I < O use 'P'E; I > O use 'N'E.

There are three possible I/O relationships for an intermediate branch, when: O>I use 'P'E and 'N'D; O∈ I use 'P'D and 'N'D; O<I use 'P'D and 'N'E. The simplest OPF containing all three intermediate branch combinations is the radix 5 Base -1 Complementer, (inverter), shown in figure 3c.

2.5: Power Supply Voltages and Designators

While the power supply voltage limits are determined by the specifications of the FETs employed, the suggested minimum supply voltages for logic levels 0 and 1 are 0.0 volts and 1.5 volts, respectively. And each additional logic level be the previous logic level voltage plus the logic level 1 voltage to provide a Logic Step Voltage (LSV) of 1.5 volts.

In schematics the power supply designators are simply the letter "V" (voltage) subscripted with the logic level represented by that voltage, (e.g. V_3 represents logic level 3).

2.6: Logic Level Determination

Due to the high input impedance of FETs and because each supply voltage represents only one logic level, the logic levels of SUS-LOC based circuits are, in virtue, equal to the supply voltages representing the logic levels. Section 2.7 establishes a Domain for each of the 'r' different logic levels.

2.7: Threshold Voltage Determination

The $V_{GS(TH)}$ of P-channel FETs is set at a percentage of the LSV above the highest input logic level to which they are to conduct. The $V_{GS(TH)}$ of N-channel FETs is set at a percentage of the LSV below the lowest input logic level to which they are to conduct.

This percentage of the LSV, at which the $V_{GS(TH)}$ of the FETs is set, is called the Overlap Percentage (OP) and has a suggested range of 55% to 75% of the LSV, such that an overlap of 'ON' branches is obtained when the circuit is switching from one output logic level to another. It is suggested that the OP be the same for all switches in digital applications, analog applications may require that the $V_{GS(TH)}$, OP, and/or the LSV be variable.

When used as suggested the OP maintains symmetry, enhances the transfer characteristics, and creates a Domain for each logic level.

To calculate the $V_{GS(TH)}$ for a particular FET, use the following equations selected for the FET's channel type, where: Vi is the input logic level voltage limit responded to, and Vo is the required output logic level voltage.

P-channel:

$$V_{GS(TH)} = Vi - (Vo - (OP \times LSV))$$

N-channel:

$$V_{GS(TH)} = Vi - (Vo + (OP \times LSV))$$

3.0: Example Functions

Because those skilled with binary logic are primarily concerned with the OPF called an Inverter, as well as the NOR, NAND, and XOR multiple place functions, the analogous ternary functions are shown and briefly discussed.

The following parameters, (selected for simplicity), were used to develop the ternary circuits shown and discussed:

V_2 = 5.0 volts LSV = 2.5 volts
V_1 = 2.5 volts OP = 70%
V_0 = 0.0 volts

3.1: Ternary Base -1 Complementer (Inverter)

The ternary Base -1 Complementer, or f(x) = <210>, is analogous to the binary inverter. The ternary Base -1 Complementer schematic and its symbol are shown in figures 2a and 2b respectively.

SPICE simulation of the chain of ten ternary Base -1 Complementers shown in figure 2c, using 1 micron transistor models, produced the output waveforms shown in figure 2d.

Shown in table A is the average power information from the simulations of the ternary chain of ten and a binary chain of ten invertors. The average power of the ternary chain is lower than the binary chain when the input is the ternary sequence of 0-1-2-1-0. The average power of the ternary chain is higher when the input sequence is 0-2-0, however, it is not twice the binary chain's average power although the ternary chain traverses 20 logic levels as opposed to the binary chain's 10.

TABLE A

Input	Binary	Ternary
0-2-0	29.1μW	50.5μW
0-1-2-1-0	N/A	16.69μW

3.2: Functions: f(x) = <200> and <220>

Because two ternary functions f(x)=<200> and f(x)=<220> are used symbolically in section 4.4 below, their schematics are shown in figures 3a and 3b, respectively.

4.0: Examples of Multiple Place Functions

An MPF is essentially the combination of two or more OPFs of the same radix, (MPFs with multiple radices is not within the scope of this paper). Each input of an MPF is the input to one or more of the OPFs used to form the MPF.

4.1: Complementing Generalized OR

The Complementing Generalized OR gate ($CGOR_3$), shown in figure 4, is analogous to a binary NOR gate. The output of a $CGOR_3$ gate is the Base -1 Complement of the highest input logic level presented to its inputs.

4.2: Complementing Generalized AND

The Complementing Generalized AND gate ($CGAND_3$), shown in figure 5, is analogous to a binary NAND gate, and its output logic level is the Base -1 Complement of the lowest input logic level presented to its inputs.

4.3: Combined One Place Functions

To show that an MPF is the combination of two or more one place functions of the same radix, as discussed in section 4.0 above, figures 4 and 5 are referenced. Both gates are comprised of one radix 3 Base -1 Complementer per input term. One Complementer is comprised of Q2, Q4, Q6 and Q7, while the other is comprised of Q1, Q3, Q5 and Q8.

The distinction between a $CGOR_r$ and a CGAND is defined by: 1) which composite branch, comprised of terminus branches, is connected in series, and which is connected in parallel; and 2) which channel type forms the series and parallel portions of the composite branch formed by the intermediate branches.

4.4: Complementing Exclusive Generalized OR

The Complementing Exclusive Generalized OR ($CXGOR_3$) is analogous to a binary XOR gate. And, two different implementations are presented in figures 6 and 7 to show the result of selecting different OPFs to form a gate.

Each implementation of the $CXGOR_3$ has a different speed, power requirement, and production consideration based completely on the OPFs selected to form the function.

5.0: Multiple Implementation of Functions

Because multiple implementations of a given function are possible using the SUS-LOC structure each different implementation of a function can have a different set of speed, power requirement, component count, and production consideration trade-offs. This is illustrated by the different implementations of the $CXGOR_3$ in figures 6 and 7.

To manually resolve the power, speed, component count, and production consideration trade-offs would require a time consuming examination of the implementations available.

An alternative approach is to automate the design process thereby allowing the trade-offs to be resolved programmatically as outlined in reference [2], or a modified version thereof.

The modifications would be such that the optimization of a connective, or block of logic would include both the logical function as well as the hardware characteristics of the various connective implementations available, and allow manually entered criteria. The use of programmatic methods would be an efficient approach to finding the best OPFs to form MPFs, and/or OPFs and MPFs to form a given block of logic.

Conclusion

This paper has presented the fundamentals of the SUS-LOC structure, a fully active and self sustaining architecture for the design and manufacture of MVL circuits.

The SUS-LOC structure provides circuit designers with the ability to realize redundant functionally complete sets of logic of any radix. Logic circuits based on the SUS-LOC structure feature high speed, well defined logic levels, and low power requirements, especially the quiescent power. These features, combined with the necessary fabrication technology being currently available and well known, makes the SUS-LOC structure well suited for MVL or Post Binary logic circuits.

Acknowledgments

The Author acknowledges Dr. Claudio Moraga, Dortmund University, and Dr. Wayne Current, University of California at Davis, for their suggestions and encouragement that this paper be submitted.

References

[1] The Supplementary Symmetrical Logic Circuit Structure has a patent pending status, and this paper is an excerpt thereof.

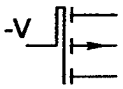
[2] C. Moraga, W. Wang: Design of Quaternary digital Circuits, Proc. 28th ISMVL, p 89-94, IEEE-CS-Press, 1998.

[3] Young-hoon Chang, Jon T. Butler: The Design of Current Mode CMOS Multiple-Valued Circuits, Proc. 21st ISMVL, p 130-138, 1991.

[4] Konrad Lei, Zvonko G. Vranesic: On the Synthesis of 4-Valued Current Mode CMOS Circuits, Proc. 21st ISMVL, p 147-155, 1991.

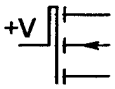
[5] Mostafa Abd-El-Barr, Muhammad Nayyar Hasan: New MVL-PLA Structures based on Current-mode CMOS Technology, Proc.26th ISMVL, P 98-103, IEEE-CS-Press, 1996.

Figure 1a:



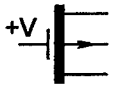
P-channel
Enhancement mode

Figure 1b:



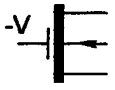
N-channel
Enhancement mode

Figure 1c:



P-channel
Depletion mode

Figure 1d:



P-channel
Depletion mode

Figure 2a:

Ternary
Base -1 Complementer
Schematic

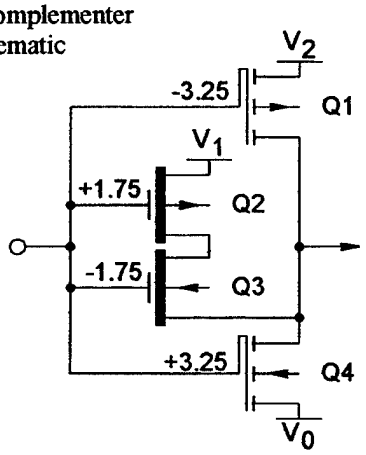


Figure 2b:

Ternary
Base -1 Complementer
Symbol

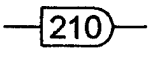


Figure 2c:

Chain of 10 Base -1 Compementers,
(R3_CHAIN10)

Figure 2d:

SPICE output waveforms of a ternary chain of ten Base -1 Complimenters

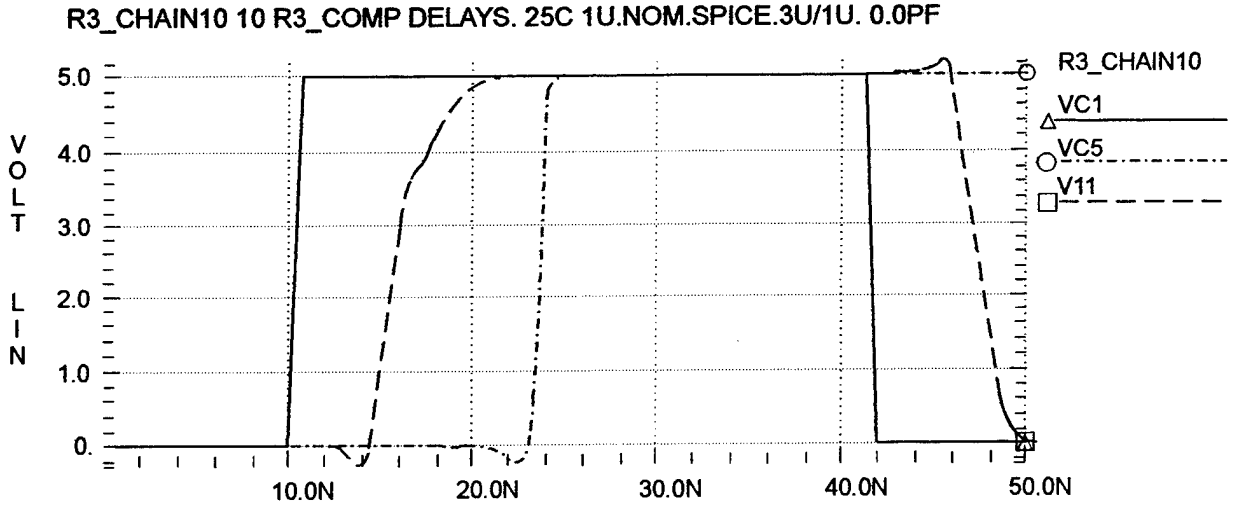


Figure 3a:

ternary f(x) = <200>

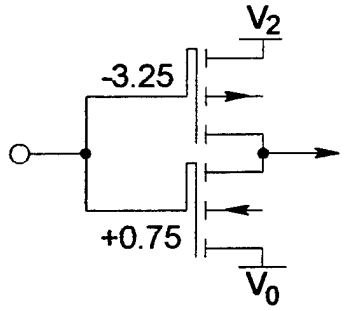


Figure 3b:

ternary f(x) = <220>

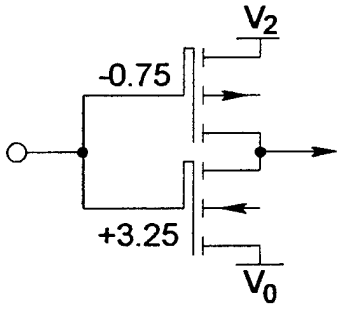


Figure 3c:

radix 5 Base -1 Complementer
f(x) = <43210>

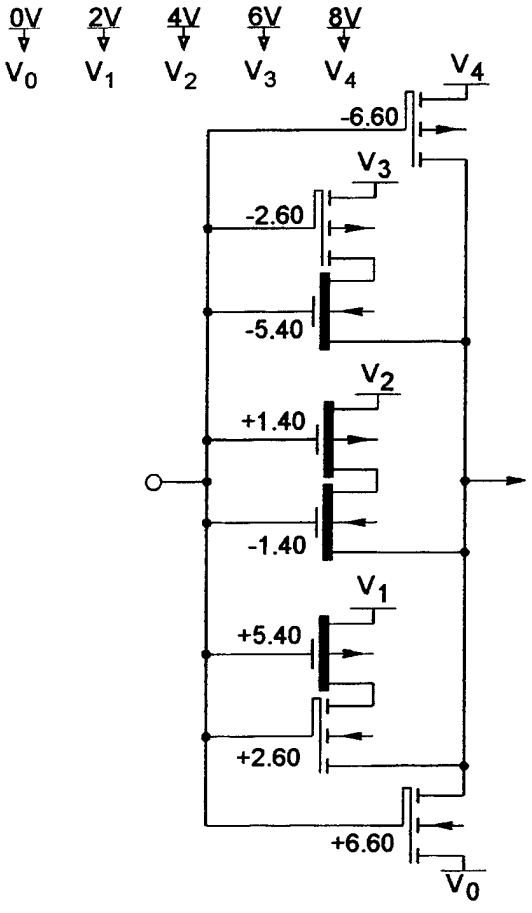


Figure 4:

CGOR 3

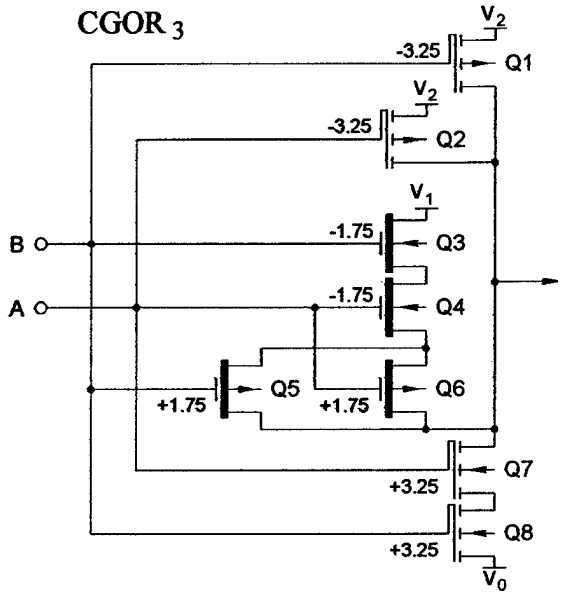


Figure 5:

CGAND 3

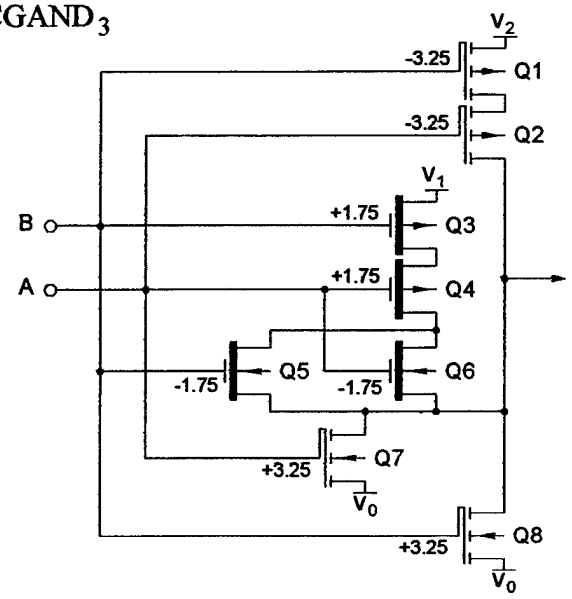


Figure 6:

CXGOR$_3$ - Implemented with f(x) = <210>

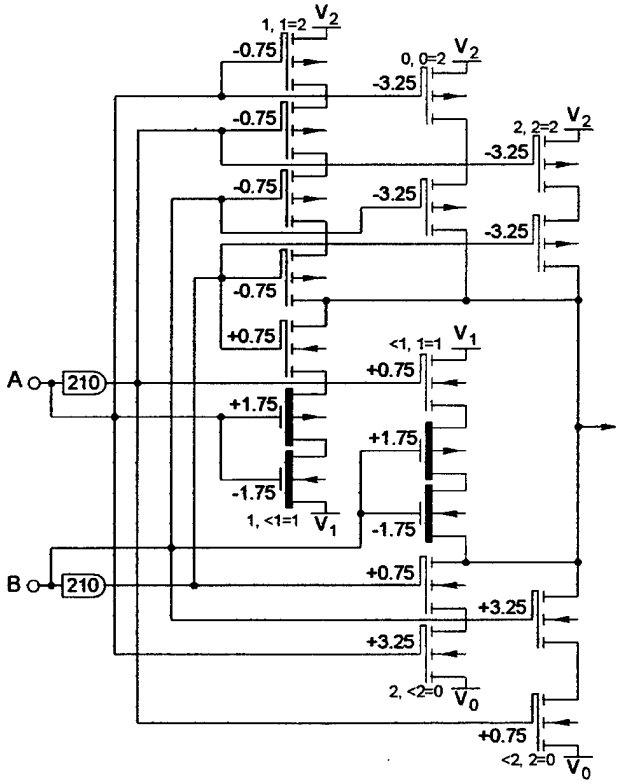
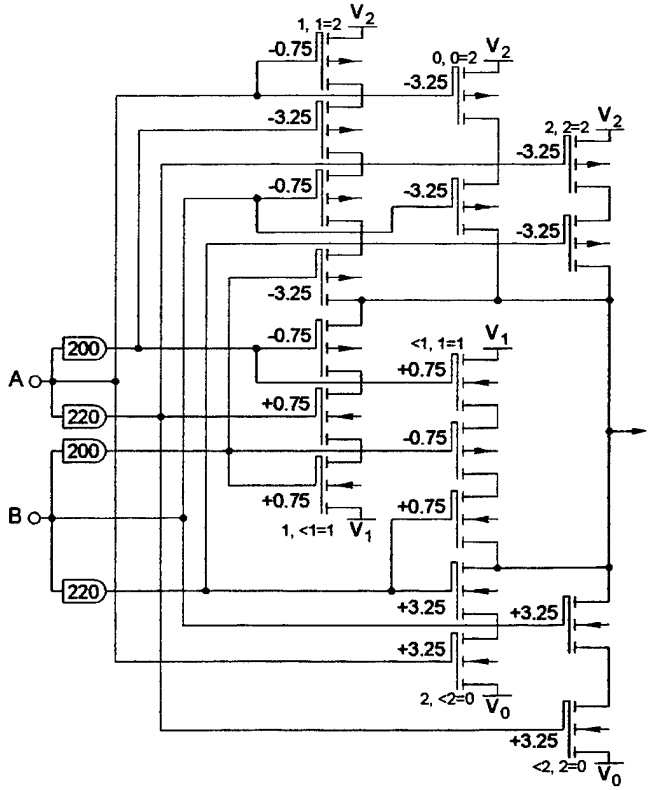
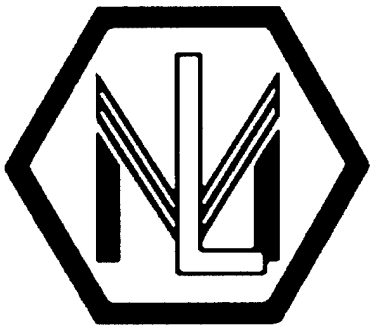


Figure 7:

CXGOR$_3$ - Implemented with f(x) = <200> and <220>



SESSION IIIA
DECOMPOSITION
CHAIR: Jon Muzio



BI-DECOMPOSITIONS OF MULTI-VALUED FUNCTIONS FOR CIRCUIT DESIGN AND DATA MINING APPLICATIONS

Bernd Steinbach, Marek A. Perkowski +, and Christian Lang,

Dept. of Computer Science, Freiberg University of Mining and Technology,

Bernhard von Cotta Strasse 1, 09596 Freiberg, Sachsen, Germany, steinb@informatik.tu-freiberg.de

+ Dept. of Electrical and Computer Engineering, Portland State University,

P.O Box 751, Portland, OR 97207-0751, USA, mperkows@ee.pdx.edu

Abstract

We present efficient algorithms for the bi-decomposition of arbitrary incompletely specified functions in variable-valued logic. Several special cases are discussed. The algorithms are especially applicable for Data Mining applications, because, in contrast to the general multi-valued approaches to function decomposition that decompose to arbitrary tables, we create a network from multi-valued two-input operators that are selected by the user. Such decompositions lead to decision rules that are easier to understand by humans.

1 Introduction

Simple Functional Decompositions.
Simple disjoint decompositions $F = H(G(B), A)$ have been used for FPGA synthesis [13], synthesis for layout-driven logic synthesis, Machine Learning and Data Mining [9, 10, 11, 6, 18, 19]. Larger block F of logic is split to smaller blocks G and H, and next blocks G and/or H are recursively split to smaller and smaller blocks, until they become non-decomposable, or until they can be directly realized with some other means (such as in a single PLA block of a CPLD). Set **B** of variables is called the *bound set*, and set **A** is called the set of *free variables*. In binary decomposition the output of function G is encoded as a binary vector of functions $G_i(B)$. In multi-valued decomposition the output of function G is a multi-valued variable. In disjoint decomposition sets **A** and **B** have no common elements. Efficient methods to represent functions in decomposition have been recently created: BDD-Encoded Labeled Rough Partitions [6], and BDD-Encoded Multi-Valued Decision Diagrams [5], but, for the simplification of presentation, in this paper we will use Kmaps to illustrate our concepts. The decomposition principle is always to find the solution with the smallest possible complexity. The complexity can correspond to the total size of non-decomposable blocks, the number of blocks, or some other evaluation of the result. This is obvious for FPGA or layout circuit applications, but it holds also for Machine Learning and Data Mining, where a simpler expression, satisfying the Occam Razor Principle, creates a more meaningful solution, and one that minimizes the learning error [4, 14]. Various decompositional approaches, mainly based on Ashenhurst and Curtis decompositions, have been realized in several programs that can handle both binary and multi-valued, completely and incompletely specified functions and relations, [11]. The methods have been also extended to non-disjoint decompositions [9, 10]. Simple non-disjoint decomposition is: $F = H(G(B_1 \cup C), A_1 \cup C)$, where **C** is the set of *shared variables*. Thus, sets **A** $= A_1 \cup$ **C** and **B** $= B_1 \cup$ **C** are *non-disjoint*.

Decomposition in Multi-Valued Circuit Design.
Turning our attention to multi-valued circuit design, let us first observe that there exist currently very few methods for synthesis of multi-level multi-valued circuits. They include *factorization methods*, *decision diagram based-methods*, and *decomposition* [3, 11]. Recent decomposers can handle data with don't cares (and multi-valued relations) better than other methods. Two approaches to multi-level multi-valued decomposition are possible. The **first approach**, *Fixed-Multiplicity Decomposition*, [3], assumes that all input and output variables, as well as all intermediate variables G_i that are created in the decomposition, have not more than a fixed constant number of values. For instance, they have at most 3 values for decomposition of ternary functions. This number is called the *multiplicity index*. The advantage of such an approach is that it allows for easier conversion of the tables describing non-decomposable blocks to Multi-Valued Sum of Products (SOP) circuits with some existing MV gates from cell libraries, or with PLA-like function generators. The **second approach**, *Unrestricted-Multiplicity Decomposition*, [11], does not assume any constraints on the number of values, and is thus better suited for Data Mining, where the data are naturally of this type because the attributes (input variables) have various sets of values. For instance, variable SEX can have as few as two values, and variable AGE can have as many as 100 values. After the decomposition of this type to tables with variables of

0-7695-0161-3/99 $10.00 © 1999 IEEE

different radii, the symbolic variables are next encoded with k-valued signals. The advantage of this approach is sometimes finding a decomposition of a smaller cost, because of a more general decomposition model used. Also, this approach allows to find decompositions that are more general than the Curtis decomposition, but are based on very similar principles [3, 9, 10]. We allow there decompositions with **arbitrary** values of multiplicity indices, which is in contrast to Curtis decomposition that in binary case requires the number μ_o of output signals to be smaller than the number μ_i of input signals to the block. In case of MV decomposition this constraint translates to **Ashenhurst decomposition** with single r-valued signal for the block or to a **Curtis-like MV decomposition** with r-valued signals from the block and less output signals than input signals to the block. Our decompositions (both binary and multi-valued) assume none of these constraints. The disadvantage of both Fixed-Multiplicity and Free-Multiplicity decompositional approaches with respect to the MV factorization and MV decision diagram approaches is that arbitrary MV blocks are created as a result of the decomposition. They are specified by tables with k-valued input and k-valued output variables. Such tables require that in order to realize the circuit corresponding to this decomposition, the blocks specified by the tables should be next either further optimized using some other MV synthesis tools, or realized as the not necessarily optimized mv-SOP circuits (using MIN, MAX and literals), that directly correspond to the non-decomposable blocks. From the point of view of circuit realizability of blocks, it would be better to decompose the function to only few selected types of blocks (MV gates), described by multi-valued operators that are directly realizable in hardware. For instance, it can be shown that every function is decomposable to *MV inverters* or *window functions*, and two-input MIN (minimum) blocks. Such realizations, however, may be very far from the minimum. (The ternary MV inverters add modulo-3 constant 1 or 2 to the value. The window literal is defined as follows: $X^i = 2$ for $X{=}i$ and 0 otherwise. These types of single-input functions can be easily extended to any number of values and realized in hardware).

Bi-Decompositions.

Our goal here is to present decompositional approaches that **decompose to finite number of selected block types**, and we will achieve this by the generalization of the bi-decomposition methods introduced originally by Davio and next discussed by many authors: Bochmann [20, 21], Le [22], Steinbach and co-authors [26]-citesteinbach-end, Zakrevskij [36, 37, 38], and Sasao/Butler [12]. The bi-decomposition approach for Boolean logic is based on AND, OR and EXOR

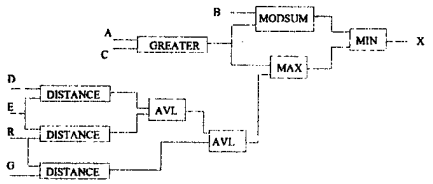


Figure 1. *Netlist with disjoint and non-disjoint bi-decompositions*

decompositions. It decomposes hierarchically an arbitrary function to three types of gates: AND, OR and EXOR. Each type of one-level decomposition, **AND-decomposition, OR-decomposition** or **EXOR-decomposition** can be disjoint or non-disjoint. Every function was proved to be decomposable, when non-disjoint decompositions are also allowed [26]. Observe, that generalizing the bi-decomposition approach to multi-valued logic would allow it to become a **practical tool for multi-level multi-valued design.** Assuming non-disjoint decompositions and availability of negation or window functions plus multi-valued constants, only MIN decompositions would be sufficient (this can be proved by generalizing the proof that NAND and constants is a complete system for binary logic).

In another variant, one would use additionally the MAX gate, which would lead to an MV counterpart of AND/OR/NOT binary logic. Creating such a system is possible and is already included in the approach presented below. The advantage of such approach would be a requirement of having very few standard MV cells. However, it can be observed that the desirable possibility of finding a decomposition with no shared variables or with a small set of shared variables greatly increases with a larger set of operators. Larger set is also a good assumption for Data Mining applications, where many different kinds of relational, mv-logic, arithmetic, and language-based operators are used by knowledge engineers and human experts that create hierarchical rule-based expert systems [17]. The question thus remains, how many gate types would be needed to obtain satisfactory solutions, because too large a number of them would be not practical. Obviously, there should be operators realizing known operators MIN, MAX, MODSUM, AVERAGE, MODULO MULTIPLICATION, TRUNCATED SUM and other. Example of a netlist being a Directed Acyclic Graph of non-disjoint and disjoint single-level bi-decompositions with operators MIN, MODSUM, MAX, GREATER, AVERAGE LOW and DISTANCE is shown in Figure 1.

It was proven by Shannon [15] that when the number of input variables n grows, the ratio of disjointly decom-

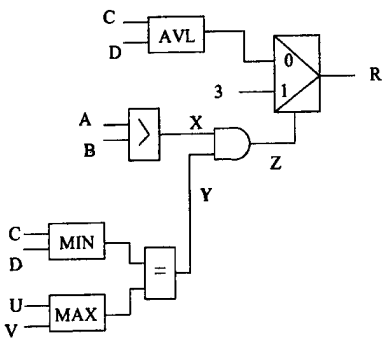


Figure 2. *Netlist from variable-valued operators corresponding to the mixed binary-MV decision rule IF* $(A < B)$ *AND* $MIN(C,D) = MAX(U,V)$ *THEN* $R = 3$ *ELSE* $R = AVERAGE(C,D)$. *Intermediate variables* X, Y *and* Z *are binary. Block of MV multiplexer is internally built from non-disjoint decomposition to MAX, two MIN, and two level-changing inverters/buffers.*

posable functions decreases to zero. From the practical point of view, however, this result is not particularly restrictive, because:

(1) Real-life functions are usually decomposable, in contrast to randomly generated worst case functions used in mathematical proofs as one from [15]. For instance, for binary functions it was shown experimentally on large benchmark sets and on many types of real-life benchmarks from different application areas that 82% of them were decomposable, while only 1% of random functions were decomposable [14].

(2) We consider non-disjoint decompositions, for which it was proven that every function is decomposable, and practically with only **few repeated variables** (disjoint decompositions are still preferable). If disjoint decompositions are not possible, we select small sets of shared variables.

(3) Functions with don't cares are better decomposable: the more don't cares, the more probable to find a disjoint decomposition. It is well-known that Data Mining functions have extremely high percent of don't cares. As demonstrated in [11], introducing of shared (the same as repeated) variables has the same effect as dealing with incomplete functions; one repeated binary variable causes creating a new Kmap with half of don't cares.

Bi-Decompositions for Data Mining.

Let us now turn our attention to a non-disjont, operator-based, hierarchical decomposition as a new approach to Data Mining. All Machine Learning and Data Mining methods assume Occam Razor principle, which is practically realized by minimizing **global cost functions** such as Decomposed Function Cardinality (DFC) [11], total multiplicity index (Column Cardinality), or other measures [16]. It was found that there is a very high correlation between the DFC value and the learning error for most of the benchmarks [14, 4]. Another important factor taken into account when evaluating various decomposition structures with blocks is also the *"understandability"* or *"explainability"* property of the resultant set of rules (expression, network, circuit) [17, 16]. Thus, between two sets of rules created by a decomposer that have the same DFC cost, the set of rules that is easier to understand by a human expert is better. Often, the human can further improve the rules, and it was shown that systems that combine computer and human knowledge acquisition on these tasks are better than computers or humans alone [17, 16]. Moreover, expert system users, such as medical doctors or lawyers, do not want to use systems that do not provide explanations that would be comprehensible to them. As one experienced data knowledge engineer working with human experts observed: *"I have never met a doctor who would use EITHER_OR (EXOR) conjunctive in his reasoning"*. Thus, the medical doctor will be never satisfied with the diagnosis provided by an expert system based on a totally "black-box" approach (such as an artificial neural net), or even using complex rules with exotic to them operators such as EXOR or MODSUM. In addition, it was observed that in real-life expert systems the functions realized in blocks are in most cases *monotonic* [17]. Thus monotonic operators should be preferably used in decomposition, or at least given preference during the decomposition choices. Another preference may be to use *symmetric operators*. Concluding, both in MV circuit design and Data Mining applications, there is a need for a method to synthesize a strongly unspecified multi-valued function with arbitrary numbers of values in variables to the preselected set of simple two-argument operators, leading to solution rules such as one illustrated in Figure 2. The goal of this paper is to introduce an efficient method for finding bi-decompositions of strongly unspecified multiple-valued functions, for selected sets of two-input operators.

2 Patterns and Decompositions of Completely Specified Functions

Operators in Ternary Logic.

For simplicity, we will illustrate our approach using ternary logic, but all algorithms can be formulated for arbitrary variable-valued logic using Multivalued Decision Diagrams [5]. Some subset of well-known operators in ternary logic are shown in Figure 3. There are several subsets of them that are functionally com-

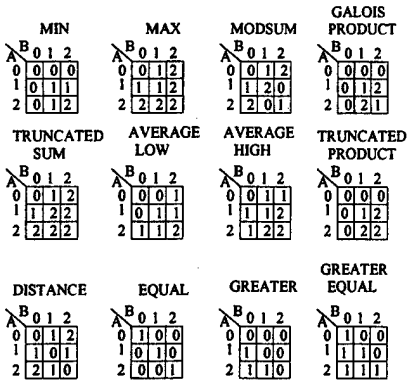


Figure 3. *Selected Operators for Ternary Logic*

↓ CD \ AB →	00	01	02	10	11	12	20	21	22
00	0	0	0	0	0	0	0	0	0
01	0	1	1	1	1	0	1	0	1
02	0	1	2	1	2	0	2	0	1
10	0	1	2	1	2	0	2	0	1
11	0	0	0	0	0	0	0	0	0
12	0	1	1	1	1	0	1	0	1
20	0	1	1	1	1	0	1	0	1
21	0	1	2	1	2	0	2	0	1
22	0	0	0	0	0	0	0	0	0

Table 1. Table for MIN decomposition of ternary function f_1

↓ CD \ AB →	00	01	02	10	11	12	20	21	22	$h(C,D)$
00	0	0	0	0	0	0	0	0	0	0
01	0	1	1	1	1	0	1	0	1	1
02	0	1	2	1	2	0	2	0	1	2
10	0	1	2	1	2	0	2	0	1	2
11	0	0	0	0	0	0	0	0	0	0
12	0	1	1	1	1	0	1	0	1	1
20	0	1	1	1	1	0	1	0	1	1
21	0	1	2	1	2	0	2	0	1	2
22	0	0	0	0	0	0	0	0	0	0
	0	1	2	1	2	0	2	0	1	$g(A,B)$

Table 2. First stage of decomposition. Functions $h(C,D)$ and $g(A,B)$ for MIN decomposition of ternary function f_1. These functions are created by labeling with symbols 0,1 and 2 identical rows and identical columns of the table, respectively

plete, with single-input window functions, inversion functions, or with arbitrary universal window functions or literals applied only in the input level (as in MV SOP realizations). Selection of functionally complete subsets is not a topic of this paper because for Data Mining larger sets of operators are better. Operators from Figure 3 are only examples, and the method shown below will lead to solutions, possibly non-optimal, for **any functionally complete set of operators.** It can be easily proven from the fundamental theorem of MV decomposition [11] that if decompositions $F = \phi(G(B), A)$ and $F = \phi(B, H(A))$ exist with multiplicity indices μ_1 and μ_2, respectively, then decomposition $F = \phi(G(B), H(A))$ exists where gate ϕ has its respective inputs with μ_1 and μ_2 values. Thus, there exists a Fixed-Multiplicity Decomposition with $MAX(\mu_1, \mu_2)$ values. For instance, if $\mu_1 = 2$ and $\mu_2 = 3$ then a ternary decomposition of a ternary function F exists.

MIN Decomposition.

To introduce our ideas, we will start with MIN Decompositions of a completely specified function. MIN decomposition has the form:

$f(X) = MIN(g(A), h(B))$.

For instance,

$f(A,B,C,D) = MIN(g(A,B),\ h(C,D))$.

Given is function $f_1(A,B,C,D)$ from Table 1. It can be observed in Table 1 that there are three row patterns and three column patterns. The patterns of rows are:

000 000 000
011 110 101
012 120 201

We say that all rows that have the same pattern are *compatible*. The patterns of columns are:

000
011
012
012
000

011
011
012
000

Columns with the same pattern are called the *compatible columns*. Table 2 presents the first stage of finding the MIN decomposition. By labeling all (compatible) rows of zeros by 0, all rows of zeros and ones by 1, and all rows of zeros, ones and 2's by 2, we obtain the additional column which describes function $h(C,D)$. By labeling all (compatible) columns of zeros by 0, all (compatible) columns of zeros and ones by 1, and all (compatible) columns of zeros, ones and 2's by 2, we obtain the additional row which describes function $g(A,B)$. From the additional row we can create directly the Kmap of function $g(A,B)$, Figure 4a. By using the permutation operation $(1-2)$ of exchanging rows 1 and 2 of variable C, Figure 4b, we obtain the map of function $h(C,D)$. This is additionally explained in Kmaps from Figure 4c-f. Finally, the entire decomposition of function f_1 can be drawn, Figure 4g. Observe that the inverter (the **1-2 permuter**) operator in variable C was used. There exist three such permuter operators (1-2, 0-1, and 0-2) for ternary logic.

Concluding, the algorithm to find ternary MIN decomposition can be summarized as follows.

[1.] Find all row patterns.

If there are more than 3 patterns, exit.

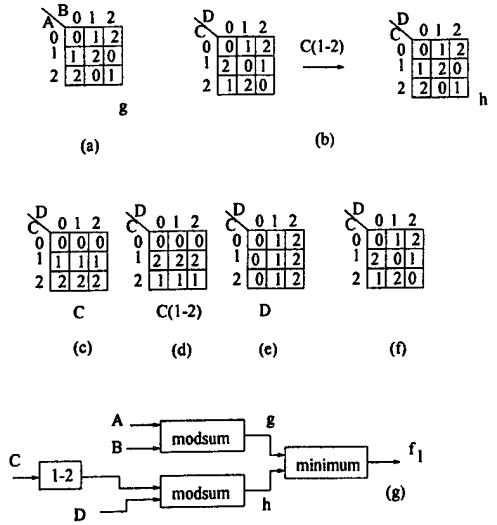


Figure 4. *Second stage of MIN decomposition of function* f_1

[2.] Find all column patterns.
If there are more than 3 patterns, exit.
[3.] Label patterns as presented above. If this is a table of MIN operator (as in Table 3), then MIN decomposition exists.

This method is a generalization of the method from [12]. Let us now explain **another possible approach** to the same problem, this approach generalizes the method from [26]. Analyze the patterns of rows column-by-column. We find that after removing all repeated columns, there are only three patterns: 000, 012 and 011. Similarly, removing all repeated patterns of rows from patterns of columns above, the remaining patterns are: 000, 012 and 011. The same patterns as before. Thus the decomposition operator from Table 3 was found which is the operator of MIN. Both the above methods will become useful, when it comes to find patterns in cofactors of large bound and free sets for incompletely specified functions.

MAX Decomposition.

Given is function f_2 from Table 4, with bound and free sets of variables $\mathbf{B} = \{A,B\}$ and $\mathbf{A} = \{C,D\}$. Following the first procedure above, Table 5, we find the realization from Figure 5 with the same functions h and g as previously. Now, following the second procedure above, but labeling the columns and rows of 0's, 1's and 2's as 0; the columns and rows of 1's and 2's as 1; and the columns and rows of 2's as 2; we found that the reduced pattern table, Table 6, is the table of operator MAX. Analogously, the same solution is found using the second approach which we leave to the Reader.

MODSUM Decomposition.

$g \setminus h$	0	1	2
0	0	0	0
1	0	1	1
2	0	1	2

Table 3. Table of ternary MIN operator for function f_1

↓ CD \ AB →	00	01	02	10	11	12	20	21	22
00	0	1	2	1	2	0	2	0	1
01	1	1	2	1	2	1	2	1	1
02	2	2	2	2	2	2	2	2	2
10	2	2	2	2	2	2	2	2	2
11	0	1	2	1	2	0	2	0	1
12	1	1	2	1	2	1	2	1	1
20	1	1	2	1	2	1	2	1	1
21	2	2	2	2	2	2	2	2	2
22	0	1	2	1	2	0	2	0	1

Table 4. Table for MAX decomposition of ternary function f_2

↓ CD \ AB →	00	01	02	10	11	12	20	21	22	$h(C,D)$
00	0	1	2	1	2	0	2	0	1	0
01	1	1	2	1	2	1	2	1	1	1
02	2	2	2	2	2	2	2	2	2	2
10	2	2	2	2	2	2	2	2	2	2
11	0	1	2	1	2	0	2	0	1	0
12	1	1	2	1	2	1	2	1	1	1
20	1	1	2	1	2	1	2	1	1	1
21	2	2	2	2	2	2	2	2	2	2
22	0	1	2	1	2	0	2	0	1	0
	0	1	2	1	2	0	2	0	1	$g(A,B)$

Table 5. Table for functions $h(C,D)$ **and** $g(A,B)$ **for MAX decomposition of ternary function** f_2**. These functions were calculated analogously as in Table 2**

	0	1	2
0	0	1	2
1	1	1	2
2	2	2	2

Table 6. Table of MAX operator

↓ CD \ AB →	00	01	02	10	11	12	20	21	22
00	0	1	2	1	2	0	2	0	1
01	1	2	0	2	0	1	0	1	2
02	2	0	1	0	1	2	1	2	0
10	2	0	1	0	1	2	1	2	0
11	0	1	2	1	2	0	2	0	1
12	1	2	0	2	0	1	0	1	2
20	1	2	0	2	0	1	0	1	2
21	2	0	1	0	1	2	1	2	0
22	0	1	2	1	2	0	2	0	1

Table 7. Table for MODSUM decomposition of ternary function f_3

	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

Table 8. Table of ternary MODSUM operator

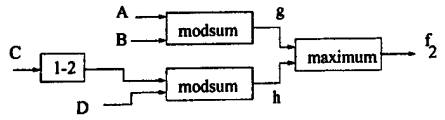


Figure 5. *MAX decomposition for function* f_2

↓ CD \ AB →	00	01	02	10	11	12	20	21	22
00	0	0	0	0	0	0	0	0	0
01	1	1	1	1	1	1	1	1	1
02	2	2	2	2	2	2	2	2	2
10	2	2	2	2	2	2	2	2	2
11	0	0	0	0	0	0	0	0	0
12	1	1	1	1	1	1	1	1	1
20	1	1	1	1	1	1	1	1	1
21	2	2	2	2	2	2	2	2	2
22	0	0	0	0	0	0	0	0	0

Table 9. Table of function *temp* for MODSUM Decomposition of function f_3

Given is function f_3 from Table 7, with bound set of variables **B** = {A,B} and free set of variables **A** = {C,D}. Following the same two procedures as previously, we find that decomposition $f_3 = MODSUM(g(A,B),\ h(C,D))$, with the same functions $h(C,D)$ and $g(A,B)$ as in the two previous examples, and the reduced pattern table, Table 8 is the table of operator MODSUM.

In this case, **one more approach** is possible:

(1) calculate function g: first row of f_3: 012120201 g.

(2) calculate a temporary function

$temp = Mod_diff(f_3(A,B,C,D),\ g(A,B))$.

(3) calculate function h: first column of f (constant values in each row are neccessary for MODSUM decomposition)

CD	h
00	0
01	1
02	2
10	2
11	0
12	1
20	1
21	2
22	0

Functions g and h can be found as previously and the operator table of the decomposition operator is MODSUM, see Table 8.

Composing two-input operators from other operators.

Even if a decomposition with an unknown operator is found using the method of compatible columns, this operator can be still composed from the known operators.

Example: Function f_4 from table in Figure 6 cannot be decomposed by MIN(A,B), MAX(A,B) or MODSUM(A,B). But $f_4 = MAX(g,h)$, where functions g and h are as in Figure 6b,c. Thus, $f_4 = MAX(MODSUM(011,002), MODSUM(002,011))$ see Figure 6d, where $< 011 >$ and $< 002 >$ are universal literals.

Conclusion on these examples.

The above three examples suggest two general procedures for a completely specified function. The **first**

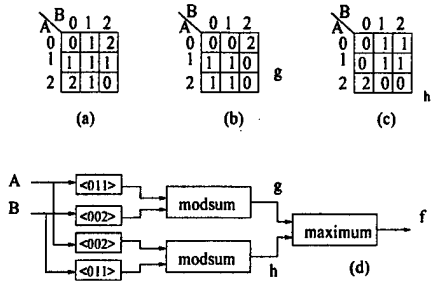


Figure 6. *Schematic of composed realization of an operator using MAX decomposition for function f_4*

procedure, that we will call the *Compatibility Algorithm*, is based on labeling arbitrary compatible patterns, calculating multiplicity indices, and next finding tables of functions F, g and h. This procedure is the same for any two-input decomposing operator, it does not assume any specific operator, and the operator becomes known only when the decomposition is completed. Thus, the user has no possibility to influence the decomposition type. This may lead to an operator that is not one of the selected operators. On the other hand, this method allows for permuting inputs, thus each operator pattern corresponds to one of many possible compositions of two-input and single input operators, so it is efficient. In case of incompletely specified function, the Compatibility Algorithm would require, however, coloring of two large *incompatibility graphs*; one with nodes for the cofactors of the bound set, and the other one with nodes for the cofactors of the free set [10]. The graphs are large, because we split the set of input variables into two sets of approximately the same size. If for both of these graphs we are able to find coloring with 3 or less colors (multiplicity index 3 or less), the binate decomposition exists. In contrast to the 2-Colorability, which is polynomial and for which we have implemented an efficient algorithm [8], the 3-Colorability Problem is NP-complete. To complicate things more, because the minimal graph coloring is usually not unique, several decompositions can be found, with various decomposition operators. Such coloring-based procedure would be not very efficient, and would not give preferences to the desired decomposition types such as monotonic, or symmetric. The **second approach**, called *Operator Pattern Algorithm*, assumes certain type(s) of decomposing operator, and looks for a decomposition only with respect to this operator. It is thus an *operator-oriented decomposition*. Therefore below, for incomplete functions, we will present a more efficient and comprehensive approach, that will make use of specific properties of the bi-decomposition of multi-

valued incomplete functions, and will combine the ideas from both the algorithms.

3 Decompositions of Incompletely Specified Functions

Number of Decompositions.
In case of a k-valued logic and n variables, there are k^{k^n} functions. If we combine two of these functions, one for rows and one for columns, we get $k^{k^n} * k^{k^n} = k^{k^n+k^n}$ functions realizable by decomposition. This number is much smaller than the number of all possible functions of $2 * n$ variables: $k^{k^{2n}}$ functions. Taking thus an arbitrary randomly selected function of n variables, we have a probability of $1 - \frac{k^{2*k^n}}{k^{k^{2n}}}$ that this function is **not** decomposable. Thus random complete functions of many variables are very unlikely to be disjoint-decomposable. Recall, however, that an incomplete function of $2n$ variables specified on K cares corresponds to $k^{k^{2n}-K}$ complete functions. Thus taking an arbitrary function with K cares, we have the probability of $(1 - \frac{k^{2*k^n}}{k^{k^{2n}}})^{k^{k^{2n}-K}}$ that this function is not decomposable. When K/k^{2n} is small, as in Data Mining where it can be less than 0.01 percent, the probability of finding disjoint decomposition is then quite high, even for random functions. Still, if the disjoint decomposition does not exist, addition of every shared variable improves the chance of finding a decomposition. The procedure, however, may become inefficient, so good methods for finding bound and free sets are needed, as well as a fast algorithm for basic decomposition execution step (because it is repeated very many times).

MIN, MAX and Simple Monotonic Decompositions.
MIN Decomposition of an incompletely specified function is based on the same principles as illustrated in previous sections for complete functions. Simplified algorithm for MIN Decomposition is the following:
[1] Find rows and columns with pattern of only 0's and don't cares, label respective rows and columns by 0. Remove 0's from these rows and columns.
[2] Find rows and columns with compatible patterns of 1's and don't cares, label respective rows and columns by 1. Remove 1's from these rows and columns. If there is more than one compatible pattern of 0's and 1's, exit.
[3] Find rows with compatible patterns of 2's and don't cares, label respective rows and columns by 2. Remove 2's from these rows and columns. If some patterns remain, exit.
[4] The labeled rows and columns determine the MIN operator pattern.

To understand this algorithm the Reader is advised to apply it to our previous tables from MIN decomposition. Also, try to apply it to the MIN table from Figure 3. Observe that similar algorithm can be created for MAX decomposition, but the order of removals will be not [0,1,2] as for MIN presented above, but [2,1,0]. This can be checked on the MAX table from Figure 3. Similarly it can be checked in Figure 3 that TRUNCATED-PRODUCT operator (TRUNCATED-PRODUCT decomposition) will be found for order [0,2,1], GREATER operator for order [0,1,0], and GREATER-EQUAL operator for order [1,0,1]. All such functions, for which the decomposition can be checked by the parametrized variants of the above algorithm with different orders of removals, we will call the *Simple Monotonic Functions.* Functions for which the removal can be done partially are also good candidates for non-disjoint decompositions, and we call them the *Partially Monotonic Functions.* In tables from Figure 3, GALOIS-PRODUCT and TRUNCATED-SUM are such functions.

Preprocessing for graph coloring.
Because graph coloring [8, 9, 11] can be slow, we propose to apply the following algorithm that in many cases removes totally the necessity of coloring, and in other cases reduces significantly the size of the rows and column incompatibility graphs.
[1] Combine all rows that have the same overlapping symbols.
[2] Combine all columns that have the same overlapping symbols.
[3] Iterate until no rows or columns can be combined.
[4] If the size of the resultant matrix is 3 x 3, bi-decomposition is found. The decomposition operator is specified by the matrix. This matrix can have don't cares, thus specifying various decomposition operators. Moreover, various functions g and h can be created from the traces of the combined (i.e., labeled the same way) rows or columns in the process of combining. If more than 3 rows or columns were found that cannot be combined to 3 patterns, exit.
[5] Execute two Graph Colorings for functions g and h. For each, if the chromatic number is larger than 3, exit. Otherwise combine in each the nodes colored with the same color creating the operator table.
Example. Given is the table of function $f_6(A, B, C, D)$ from Table 10. By combining rows table 11 is created, and next by combining columns in it, Table 12. Observe that by combining first columns and next rows, Tables 13, 14, another solution is found. Observe in Table 14 the final matrix specifying the ternary decomposition operator. Note, the decomposition is different than in Table 12 because row labels are different. Because of a don't care in the operator table, it corresponds to three different decomposition operators; with the don't care replaced with 0, with 1, or with 2. Thus, several solutions may be found, and the one correspond-

ing to the realizable decomposition operator is next chosen.

4 Experimental Results and Conclusions

Table 15 gives results of bi-decomposition of some Benchmark functions from POLO directory [4, 11, 16] (Data Mining and Machine Learning). We tried decomposition for the MIN and MAX operator by successive removal of values as described in section 3 for simple monotonic functions. That is, function $f(xa, xb, xc) = MIN/MAX(g(xa, xc), h(xb, xc))$. The meaning of the columns is: $Name$ = Name of the Benchmark, $\#In$ = Number of multi-valued inputs, $log(In) = log_2$(Product of multiplicity indices of inputs), Out = multiplicity index of output, $\#GN$ = number of (multi-valued) inputs of g for MIN, $\#HN$ = number of (multi-valued) inputs of h for MIN, $\#GX$ = number of (multi-valued) inputs of g for MAX, $\#HX$ = number of (multi-valued) inputs of h for MAX. The dashes mean that there is no decomposition. The total computation time was 10 minutes for all Benchmarks of the table on a 133 MHz Pentium Processor.

We generalized the binary bi-decompositions discussed previously by Steinbach and Sasao/Butler for the case of **arbitrary variable-valued logic**. Although bi-decomposition is only a special case that can be derived from the general-purpose Ashenhurst/Curtis decomposition, this decomposition is especially important in Knowledge Discovery, Data Mining, and automatic knowledge acquisition applications, because **it creates networks that when converted to sets of rules, are easier to understand.** Moreover, although every function is disjointly or non-disjointly decomposable this way, the MV bi-decompositions are especially efficient and effective for the case of **strongly unspecified** functions, characteristic for Data Mining.

↓ CD \ AB →	00	01	02	10	11	12	$h(C, D)$ label
00	0	-	0	-	1	-	a
01	0	-	-	1	1	-	b
02	-	-	0	-	1	-	c
10	0	-	0	-	-	-	d
11	0	0	-	-	-	-	e
12	0	-	-	-	-	0	f
20	0	2	2	1	1	0	g
21	0	2	2	-	1	0	h
22	-	-	2	-	1	-	i
	k	l	m	n	o	p	$g(A, B)$ label

Table 10. Table for incomplete function f_6 with labeled rows and columns

						row labels ↓
0	-	0	1	1	-	a,b,c
0	0	0	-	-	0	d,e,f
0	2	2	1	1	0	g,h,i
k	l	m	n	o	p	← column labels

Table 11. Combined rows for function from Table 10

			row labels ↓
0	0	1	a,b,c
0	0	-	d,e,f
0	2	1	g,h,i
k,p	l,m	n,o	← column labels

Table 12. Combined columns for function from Table 11, the final matrix specifying the ternary decomposition operator

			row labels ↓
0	0	1	a
0	-	1	b
-	0	1	c
0	0	-	d
0	0	-	e
0	-	-	f
0	2	1	g
0	2	1	h
-	2	1	i
k,p	l,m	n,o	← column labels

Table 13. Combined columns for function from Table 10

			row labels ↓
0	0	1	a,c
0	0	-	d,e
0	2	1	b,f,g,h,i
k,p	l,m	n,o	← column labels

Table 14. Combined rows for function from Table 13

Name	#In	log(In)	Out	#GN	#HN	#GX	#HX
lenseesmv.ml	4	5	3	3	3	3	1
shuttlem.ml	6	8	2	4	4	5	1
ships.ml	4	8	4	3	1	3	1
hayes.ml	4	9	3	-	-	-	-
iris.ml	4	12	3	3	2	3	3
post-operative.ml	8	12	3	6	5	6	5
cloud.ml	6	16	2	4	3	4	3
monks3tr.ml	6	9	2	4	2	4	2
monks1tr.ml	6	9	2	3	3	3	3
bridges1.ml	9	15	7	7	4	7	6
monks2tr.ml	6	9	2	-	-	-	-
bridges2.ml	10	17	7	7	7	7	5
soo.ml	16	19	7	8	8	8	8
irish.ml	4	9	3	2	2	2	2
monks1te.ml	6	9	2	3	3	3	3
monks2te.ml	6	9	2	3	3	3	3
monks3te.ml	6	9	2	3	3	3	3
balance.ml	4	10	3	-	-	-	-
sensory.ml	11	18	11	6	5	6	5
house-votes-84.ml	16	16	2	14	14	-	-
tic-tac-toe.ml	9	15	2	7	7	7	7
flare1.ml	10	16	2	5	5	6	5
car.ml	6	11	4	-	-	-	-
flare2.ml	10	16	3	5	5	5	5
employ1.ml	9	14	4	5	4	-	-
nursery.ml	8	14	5	-	-	-	-
employ2.ml	7	15	4	5	5	-	-
chess1.ml	6	16	18	-	-	-	-

Table 15. Experimental Results

References

[1] R. L. Ashenhurst, "The decomposition of switching functions," *Proc. Int. Symp. Th. Swi.*, pp. 74-116, April 1957.

[2] H. A. Curtis, "A New Approach to the Design of Switching Circuits," Princeton, N.J.: *Van Nostrand*, 1962.

[3] C. Files, R. Drechsler, and M. Perkowski, "Functional Decomposition of MVL Functions using Multi-Valued Decision Diagrams," *Proc. ISMVL'97*, pp. 27-32, 1997.

[4] C. Files, and M. Perkowski, "An Error Reducing Approach to Machine Learning using Multi-Valued Functional Decomposition," *Proc. ISMVL'97*, pp. 27-32, 1998.

[5] C. Files and M. Perkowski, "Implementing Multi-Valued Decision Diagram Package using Binary Decision Diagrams," *submitted.*

[6] S. Grygiel, M. Perkowski, M. Marek-Sadowska, T. Luba, and L. Jozwiak, "Cube Diagram Bundles, A New Representation of Strongly Unspecified Multiple-Valued Functions and Relations," *Proc. ISMVL'97*, May 1997, pp. 287 - 292. http://www.ee.pdx.edu/ mperkows/ML/=xxx.ps

[7] Y-T. Lai, M. Pedram, S. B. K. Vrudhula, "EVBDD-based algorithm for integer linear programming, spectral transformation, and functional decomposition," *IEEE Trans. CAD,* Vol. 13, No. 8. Aug. 1994, pp. 959-975.

[8] M. A. Perkowski, "A New Representation of Strongly Unspecified Switching Functions and Its Application to Multi-Level AND/OR/EXOR Synthesis," *Proc. RM'95,* 27-29 Aug. 1995, pp. 143-151.

[9] M.A. Perkowski, S. Grygiel, and the Functional Decomposition Group, Department of Electrical Engineering, "A Survey of Literature on Function Decomposition," Version IV, *PSU ECE Dept. Report,* Nov. 20, 1995.

[10] M.A. Perkowski, T. Luba, S. Grygiel, P. Burkey, M. Burns, N. Iliev, M. Kolsteren, R. Lisanke, R. Malvi, Z. Wang, H. Wu, F. Yang, S. Zhou, and J.S. Zhang, " Unified Approach to Functional Decompositions of Switching Functions," *PSU Electr. Engn. Dept. Report,* Dec. 29, 1995.

[11] M. Perkowski, M. Marek-Sadowska, L. Jozwiak, T. Luba, S. Grygiel, M. Nowicka, R. Malvi, Z. Wang, and J. S. Zhang, "Decomposition of Multiple-Valued Relations," *Proc. ISMVL'97,* Halifax, Nova Scotia, Canada, May 1997, pp. 13 - 18. http://www.ee.pdx.edu/ mperkows/ML/=p33.ps.

[12] T. Sasao and J. Butler, "On Bi-Decompositions of Logic Functions," *Proc. Intern. Workshop on Logic Synthesis,* Lake Tahoe, California, May 18-21, 1997.

[13] W. Wan, and M. A. Perkowski, "A New Approach to the Decomposition of Incompletely Specified Functions based on Graph-Coloring and Local Transformations and Its Application to FPGA Mapping," *Proc. EURO-DAC '92,* pp. 230 - 235, Sept. 7-10, Hamburg, 1992. http://www.ee.pdx.edu/ mperkows/ML/pap.ps.

[14] T.D. Ross, M.J. Noviskey, T.N. Taylor, D.A. Gadd, "Pattern Theory: An Engineering Paradigm for Algorithm Design," *Final Technical Report WL-TR-91-1060, Wright Laboratories,* USAF, WL/AART/WPAFB, OH 45433-6543, August 1991.

[15] C. E. Shannon, "The synthesis of two-terminal switching circuits," *Bell Syst. Techn. J.*, Vol. 28, 1949, pp. 59-98.

[16] B. Zupan, M. Bohanec, J. Demsar, and I. Bratko, "Feature Transformation by Function Decomposition," *IEEE Expert,* Spec. Issue on Feature Transformation and Subset Selection.

[17] B. Zupan, M. Bohanec, I. Bratko, B. Cestnik, "A Dataset decomposition approach to data mining and machine discovery," *preprint from authors,* 1998.

[18] M. Burns, M. Perkowski, L. Jozwiak, "An Efficient Approach to Decomposition of Multi-Output Boolean Functions with Large Sets of Bound Variables," *Proc. 1998 Euromicro,* pp. 16 - 23.

[19] R. Malvi, M. Perkowski, and L. Jozwiak, "Exact Graph Coloring for Functional Decomposition: Do we Need it?," *Proc. 3rd Int. Work. Boolean Problems,* 1998, pp. 1 - 10.

[20] D. Bochmann, and B. Steinbach, "Logikentwurf mit XBOOLE," *Verlag Technik,* Berlin 1991.

[21] D. Bochmann, F. Dresig, and B. Steinbach, "A New Decomposition Method for Multilevel Circuit Design," *European Design Automation Conference,* Amsterdam, 1991, pp. 374 - 377.

[22] T.Q. Le, and B. Steinbach, "Effiziente Methoden zum Logikentwurf testbarer kombinatorischer Schaltungen und deren impliziten Testsatzberechnung," *Proc. 3rd ITG/GI-Workshop: "Testmethoden und Zuverlassigkeit von Schaltungen und Systemen",* Blomberg, 1991.

[23] B. Steinbach, and T.Q. Le, "Entwurf testbarer Schaltnetzwerke," *Wissenschaftliche Schriftenreihe der TU Chemnitz,* H. 12/1990.

[24] B. Steinbach, "Auflosbarkeit und Eindeutigkeit Boolescher Gleichungen," *Wissenschaftliche Schriftenreihe der TU Chemnitz-Zwickau,* H. 7/1992.

[25] B. Steinbach, "XBOOLE - A Toolbox for Modelling, Simulation, and Analysis of Large Digital Systems," *System Analysis and Modelling Simulation,* Gordon & Breach Science Publishers, 9(1992), No. 4, pp. 297 - 312.

[26] B. Steinbach, F. Schumann, and M. Stoeckert, "Functional Decomposition of Speed Optimized Circuits," *in: Auvergne, D.; Hartenstein, R.: Power and Timing Modelling for Performance of Integrated Circuits,* IT Press Verlag, Bruchsal, 1993, pp. 65 - 77.

[27] B. Steinbach, and M. Stoeckert, "Design of Fully Testable Circuits by Functional Decomposition and Implicit Test Pattern Generation," *Proc. 12th IEEE VLSI Test Symposium,* Cherry Hill, New Jersey, 1994, pp. 22 - 27.

[28] B. Steinbach, and Th. Muller, "Dekompositorische Schaltungssynthese mit komplexen Logikmoduln," *Tagungunterlagen des GI/ITG - Workshops "Anwender- programmierbare Schaltungen",* Karlsruhe, 1994.

[29] B. Steinbach, and A. Wereszczynski, "Synthesis of Multi-Level Circuits Using EXOR-Gates," *Proc. RM'95,* Chiba (Makuhari), Japan, 1995, pp. 161 - 168.

[30] B. Steinbach, and K. Hesse, "Design of large digital circuits utilizing functional and structural properties," *in: Steinbach, B. (Hrsg.): Boolesche Probleme, Proceedings des 2. Workshops,* 19. und 20. September 1996, TU Bergakademie Freiberg, pp. 23 - 30.

[31] B. Steinbach, and Z. Zhang, "Synthesis for Full Testability of Large Partitioned Combinational Circuits," *in: Steinbach, B. (Hrsg.): Boolesche Probleme, Proc. 2nd Workshop,* Sept. 19-20, TU Bergakademie Freiberg, pp. 31 - 38.

[32] B. Steinbach, and Z. Zhang, "Designing for Testability of Long Pipeline of Modules," in: Anheier, W.; (ed): *Tagungsband - 9. Workshop "Testmethoden und Zuverlassigkeit von Schaltungen und Systemen",* Report 1/97, Berichte Elektrotechnik, Universitaet Bremen, 1997, pp. 74 - 77.

[33] B. Steinbach, and Z. Zhang, "Synthesis for Full Testability of Partitioned Combinational Circuits Using Boolean Differential Calculus," *in: Proc. IWLS'97,* Granlibakken Resort - Tahoe City, CA - USA, 1997, pp. 1 - 4.

[34] B. Steinbach, Z. Zhang, and Ch. Lang, "Logical Design of Fully Testable Large Circuits by Decomposition," *in: Proc. Second Intern. Conf. on Computer-Aided Design of Discrete Devices,* (CAD DD'97), Vol. 1, pp. 7 - 14, Minsk, Belarus.(http://www.informatik.tu-freiberg.de/prof2/publikationen/ld_ftc_d.ps).

[35] B. Steinbach, and A. Zakrevskij, "Three Models and Some Theorems on Decomposition of Boolean Function," *in: Steinbach, B. (Hrsg.): Boolean Problems, Proc. 3rd Intern. Workshops on Boolean Problems,* Sept. 17-18, 1998, TU Bergakademie Freiberg, pp. 11 - 18.

[36] A.D. Zakrevskij, "An algorithm for decomposition of Boolean functions," *Trudy SPhTI,* is.44, 1964, Tomsk, pp. 5-16 (in Russian).

[37] A.D. Zakrevskij, "PLA decomposition over input variables," *Doklady AN B,* 1980, Vol. 24, No. 5, pp. 419-422 (in Russian).

[38] A.D. Zakrevskij, "On a special kind decomposition of weakly specified Boolean functions," *Computer-Aided Design of Discrete Devices,* pp. 36–45, Minsk, 1997.

Totally Undecomposable Functions: Applications to Efficient Multiple-Valued Decompositions

Tsutomu Sasao
Department of Computer Science and Electronics
Kyushu Institute of Technology
Iizuka 820-8502, Japan

Abstract

A function $f : P^n \rightarrow P$, $P = \{0, 1, \ldots, p-1\}$ is k-decomposable iff f can be represented as $f(X_1, X_2) = g(h_1(X_1), h_2(X_1), \ldots, h_k(X_1), X_2)$, where (X_1, X_2) is a bipartition of input variables. This paper introduces the notion of totally k-undecomposable functions. By using this concept, we can drastically reduce the search space to find k-decompositions. A systematic method to find the bipartitions of input variables that will not produce any k-decompositions is presented. By combining it to the conventional decomposition methods, we can build an efficient functional decomposition system. This method is promising to design LUT-based FPGAs.

Key words: Functional decomposition, Symmetric function, LUT-based FPGA, Multiple-valued logic function.

I Introduction

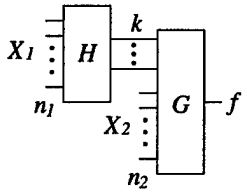
Decompositions of logic functions have been studied for many years. Major contributions are summarized as follows:

- Formulations using decomposition tables [1, 5].
- Formulations using compatibility [12].
- Fast method using Jacobian [20].
- Applications to multi-level PLA networks [13, 6].
- Extension to incompletely specified functions [21].
- Computation of column multiplicity using BDDs [14, 8, 4].
- Application to FPGAs [11].
- Bi-decomposition [15].
- Fast method [2, 10].
- Extension to multiple-valued logic [7, 9].
- Extension to multiple-output functions [18, 22, 9].

In the above contributions, most are related to two-valued functions. However, extensions to multiple-valued functions are quite natural.

In this paper, we will consider decompositions shown in Fig. 1.1. Given a multiple-valued function $f : P^n \rightarrow P$, $P = \{0, 1, \ldots, p-1\}$, we will consider the problem whether f can be represented as $f(X_1, X_2) = g(h_1(X_1), h_2(X_1), \ldots, h_k(X_1), X_2)$, or not.

Let n be the number of the input variables, then we have to consider nearly 2^n different bipartitions (X_1, X_2) of the input variables $\{x_1, x_2, \ldots, x_n\}$. When



$$f(X_1, X_2) = g(h_1(X_1), h_2(X_1), \ldots, h_k(X_1), X_2)$$

Figure 1.1: Disjoint k-decomposition.

n is large, the number of bipartitions to consider is too large, and the exhaustive search is impractical.

This paper introduces the concept of totally undecomposable functions. By using this concept, we can drastically reduce computation time to find decompositions. This paper shows a systematic method to find the bipartitions of input variables that will not produce any decompositions. By combining it to the conventional decomposition methods, we can build an efficient functional decomposition system.

The rest of this paper is organized as follows: Section II gives definitions and basic properties of functional decompositions. Section III introduces the concept of k-undecomposable functions. It also derives a theorem to find bipartitions (X_1, X_2) that will not produce any k-decomposition. Section IV shows a method to represent a set of bipartitions by using a switching function. Section V enumerates the number of k-undecomposable functions. It also shows that, for sufficiently large n, almost all functions are totally k-undecomposable.

II Definitions and Basic Properties

Definition 2.1 *A p-valued function is a mapping $f : P^n \rightarrow P$, where $P = \{0, 1, \ldots, p-1\}$ and $p \geq 2$. If $p = 2$, f is a* **switching function**.

Definition 2.2 *Let the set of the input variables be $\{X\} = \{x_1, x_2, \ldots, x_n\}$. $(X_1, X_2, \ldots, X_r)$ is a* **partition** *of X if $\{X_i\} \cap \{X_j\} = \phi$ $(1 \leq i < j \leq n)$ and $\{X_1\} \cup \{X_2\} \cup \cdots \cup \{X_r\} = \{X\}$. Especially when $r = 2$, the partition is a* **bipartition**. *The number of the variables in $\{X\}$ is denoted by $|X|$.*

Definition 2.3 *A p-valued function f has a* **disjoint k-decomposition** *iff f is represented as $f(X_1, X_2) =$*

0-7695-0161-3/99 $10.00 © 1999 IEEE

$X_2=(x_3,x_4)$ \ $X_1=(x_1,x_2)$	0 0	0 1	0 2	1 0	1 1	1 2	2 0	2 1	2 2
00	2	1	1	2	2	2	1	1	1
01	0	1	1	0	0	0	1	1	1
02	1	1	1	1	1	1	1	0	0
10	0	0	0	0	0	0	0	0	0
11	1	0	0	1	1	1	0	2	2
12	2	0	0	2	2	2	0	2	2
20	0	2	2	0	0	0	2	2	2
21	1	2	2	1	1	1	2	0	0
22	2	2	2	2	2	2	2	0	0

Figure 2.1: Decomposition table.

$g(h_1(X_1), h_2(X_1), \ldots, h_k(X_1), X_2)$, *where* (X_1, X_2) *is a bipartition of* X, *and* g *and* h_i *are p-valued functions. If* $|X_1| \geq k+1$ *and* $|X_2| \geq 1+\lceil \log_p k \rceil$, *then the decomposition is* **non-trivial**, *and* f *is* k-**decomposable**, *where* $\lceil a \rceil$ *denotes the least integer not smaller than* a. *We also assume that functions with up to two variables are decomposable.* $\{X_1\}$ *and* $\{X_2\}$ *are the* **bound set** *and the* **free set**, *respectively. Variables in* $\{X_1\}$ *and* $\{X_2\}$ *are* **bound variables** *and* **free variables**, *respectively. When* f *is* k-*decomposable,* f *is realized by the network shown in Fig. 1.1.*

Definition 2.4 *If* f *does not depend on one or more variables, then* f *is* **degenerate**.

Note that if f is degenerate, then f is decomposable.

Definition 2.5 *Let* $f(X)$ *be a p-valued function, and* (X_1, X_2) *be a bipartition of* X, *where* $n_1 = |X_1|$ *and* $n_2 = |X_2|$. *The* **decomposition table** *of* f *has* p^{n_1} *columns and* p^{n_2} *rows, each column has distinct p-ary label of* n_1 *digits, each row has distinct p-ary label of* n_2 *digits, and the corresponding entry of the table represents the value of* f.

Example 2.1 *Let* $f(X)$ *be a function* $f : \{0,1,2\}^4 \rightarrow \{0,1,2\}$, *and* (X_1, X_2) *be a bipartition of* X, *where* $X_1 = (x_1, x_2)$ *and* $X_2 = (x_3, x_4)$. *Fig. 2.1 is an example of a decomposition table.* ∎

Definition 2.6 *The number of different column patterns in the decomposition table for a bipartition* (X_1, X_2) *is the* **column multiplicity** *and is denoted by* $\mu(f : X_1, X_2)$.

Theorem 2.1 *A p-valued function* $f(X)$ *has a disjoint* k-*decomposition* $f(X) = g(h_1(X_1), h_2(X_1), \ldots, h_k(X_1), X_2)$ *iff* $\mu(f : X_1, X_2) \leq p^k$.

The size of decomposition tables for n variables is p^n, and the number of different bipartitions is $O(2^n)$. Thus, the straightforward method to find a k-decomposition is impractical for the functions with many inputs. A method to find decompositions by using ROBDDs (reduced ordered binary decision diagrams) or ROMDDs (reduced ordered multi-valued decision diagrams) has been developed.

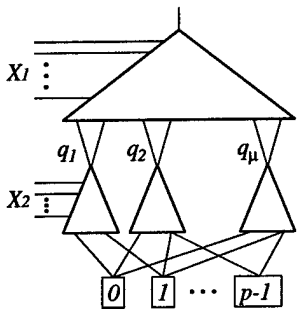


Figure 2.2: Computation of column multiplicity $\mu(f : X_1, X_2)$.

Theorem 2.2 *[14, 8, 7] Let* (X_1, X_2) *be a bipartition of* X. *Suppose that the ROMDD for* $f(X)$ *is partitioned into two blocks as shown in Fig. 2.2. The number of nodes in the lower block that are adjacent to the upper block is equal to* $\mu(f : X_1, X_2)$.

Lemma 2.1 *For any bipartition* (X_1, X_2) *of input variables of a p-valued function* f, $1 \leq \mu(f : X_1, X_2) \leq \min(p^{n_1}, p^{p^{n_2}})$, *where* $n_1 = |X_1|$ *and* $n_2 = |X_2|$.

(Proof) The number of columns in the decomposition table is p^{n_1}. Thus, we have $\mu(f : X_1, X_2) \leq p^{n_1}$. The number of different functions of n_2 variables is $p^{p^{n_2}}$. Since each column of the decomposition table shows an n_2-variable function, we have $\mu(f : X_1, X_2) \leq p^{p^{n_2}}$. □

Definition 2.7 *Let* $f(X_A, X_B)$ *be a function, where* $|X_B| = n_B$. *Let* $\vec{a}_B \in P^{n_B}$ *be an assignment for* X_B. *Then,* $f(X_A, \vec{a}_B)$ *denotes the sub-function, where the values of* X_B *are fixed to the constants* $\vec{a}_B$. $f(\vec{a}_A, X_B)$ *is similarly defined.*

Definition 2.8 *Let* $X = (x_1, x_2, \ldots, x_n)$ *and* $\vec{a} = (a_1, a_2, \ldots, a_n)$. *Then,*

$$X^{\vec{a}} \begin{cases} = p-1 \text{ if } x_i = a_i \text{ for } i = 1, 2, \ldots, n. \\ = 0 \text{ otherwise.} \end{cases}$$

Lemma 2.2 *If* $k \geq p^{|X_2|}$, *then any p-valued function is realized in the network shown in Fig. 1.1.*

(Proof) Let $n_2 = |X_2|$. An arbitrary function $f(X_1, X_2)$ is represented by $f(X_1, X_2) = \bigvee_{\vec{a} \in P^{n_2}} f(X_1, \vec{a}) X_2^{\vec{a}}$, where $P = \{0, 1, \ldots, p-1\}$. Since the number of products in the above expression is at most p^{n_2}, we have the lemma. □

Example 2.2 *When* $n_2 = 1$ *and* $k = p$, *any p-valued function is realized in the network shown in Fig. 2.3 by using the following expansion:*

$$f(X_1, X_2) = X_2^0 f_0(X_1) \vee X_2^1 f_1(X_1) \vee \cdots \vee X_2^{p-1} f_{p-1}(X_1).$$

∎

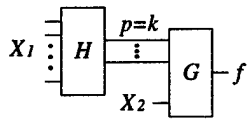


Figure 2.3: Example of a trivial k-decomposition.

	$X_1=(x_1,x_2,x_3)$							
	0	0	0	0	1	1	1	1
	0	0	1	1	0	0	1	1
$X_2=(x_4,x_5)$	0	1	0	1	0	1	0	1
00	1	1	0	0	1	0	0	0
01	1	0	0	1	0	1	1	0
10	0	0	0	1	0	1	1	0
11	0	1	1	0	0	0	0	0

Figure 3.1: Totally 2-undecomposable function.

We assume that the k-decomposition in Lemma 2.2 is trivial. This is why we assumed that $n_2 \geq \lceil \log_p k \rceil + 1$ in Definition 2.3.

III k-Undecomposable Functions

In this part, we introduce the notion of totally k-undecomposable functions. We will show that if $f(X_A, \vec{a}_B)$ is totally k-undecomposable, then $f(X_A, X_B)$ is k-undecomposable for many bipartitions. Thus, by finding totally k-undecomposable subfunctions, we can drastically reduce the search space for functional decompositions.

Definition 3.1 *A p-valued function $f(X)$ is* **totally k-undecomposable** *if $\mu(f : X_1, X_2) > p^k$ for any bipartition (X_1, X_2), where $|X_1| \geq 1 + k$ and $|X_2| \geq 1 + \lceil \log_p k \rceil$.*

Example 3.1 *Consider the case where $n = 3$, $p = 2$, and $k = 1$. $f(x_1, x_2, x_3) = x_1x_2 \vee x_2x_3 \vee x_3x_1$ is totally 1-undecomposable.* ∎

Example 3.2 *Consider the case where $n = 5$, $p = 2$, and $k = 2$. A five-variable function f shown in Fig. 3.1 is totally 2-undecomposable, since $\mu(f : X_1, X_2) > 4$ for any bipartitions with $|X_1| = 3$ and $|X_2| = 2$.* ∎

Lemma 3.1 *Let $(X_{1A}, X_{1B}, X_{2A}, X_{2B})$ be a partition of X, where $|X_{1A}| \geq k + 1$ and $|X_{2A}| \geq 1 + \lceil \log_p k \rceil$. Let $\vec{a}_{1B}$ and $\vec{a}_{2B}$ be assignments of X_{1B} and X_{2B}, respectively. If $f(X_{1A}, \vec{a}_{1B}, X_{2A}, \vec{a}_{2B})$ has no k-decomposition of the form*

$$\begin{aligned}\hat{f}(X_{1A}, X_{2A}) &= f(X_{1A}, \vec{a}_{1B}, X_{2A}, \vec{a}_{2B}) \\ &= \hat{g}(\hat{h}_1(X_{1A}), \hat{h}_2(X_{1A}), \ldots, \hat{h}_k(X_{1A}), X_{2A}),\end{aligned}$$

then, $f(X_{1A}, X_{1B}, X_{2A}, X_{2B})$ has no k-decomposition of the form

$$\begin{aligned}&f(X_{1A}, X_{1B}, X_{2A}, X_{2B}) \\ &= g(h_1(X_{1A}, X_{1B}), h_2(X_{1A}, X_{1B}), \ldots, \\ &\quad h_k(X_{1A}, X_{1B}), X_{2A}, X_{2B}).\end{aligned}$$

In this case, $\{X_{1B}\}$ or $\{X_{2B}\}$ can be empty set(s).

(Proof) Assume that f has a k-decomposition of the form

$$\begin{aligned}&f(X_{1A}, X_{1B}, X_{2A}, X_{2B}) \\ &= g(h_1(X_{1A}, X_{1B}), h_2(X_{1A}, X_{1B}), \ldots, \\ &\quad h_k(X_{1A}, X_{1B}), X_{2A}, X_{2B}).\end{aligned}$$

Assign $\vec{a}_{1B}$ and $\vec{a}_{2B}$ to X_{1B} and X_{2B}, respectively. Then, we have the decomposition $f(X_{1A}, \vec{a}_{1B}, X_{2A}, \vec{a}_{2B}) = g(h_1(X_{1A}, \vec{a}_{1B}), h_2(X_{1A}, \vec{a}_{1B}), \ldots, h_k(X_{1A}, \vec{a}_{1B}), X_{2A}, \vec{a}_{2B})$. However, this contradicts the assumption of the lemma. □

Example 3.3 *Consider the case where $n = 8$, $k = 2$, and $p = 2$. Let $f(x_1, x_2, \ldots, x_8)$ be an 8-variable function. If $(x_1, x_2, x_3, 0, 1, x_6, x_7, 1)$ has no 2-decomposition of the form $\hat{f}(x_1, x_2, x_3, x_6, x_7) = \hat{g}(\hat{h}_1(x_1, x_2, x_3), \hat{h}_2(x_1, x_2, x_3), x_6, x_7)$, then $f(x_1, x_2, \ldots, x_8)$ has no 2-decomposition of the form $f = g(h_1(x_1, x_2, x_3, x_4, x_5), h_2(x_1, x_2, x_3, x_4, x_5), x_6, x_7, x_8)$. In this example, $X_{1A} = (x_1, x_2, x_3)$, $X_{1B} = (x_4, x_5)$, $X_{2A} = (x_6, x_7)$, $X_{2B} = (x_8)$, $\vec{a}_{1B} = (0, 1)$, and $\vec{a}_{2B} = 0$.* ∎

Theorem 3.1 *Let (X_A, X_B) be a partition of X, where $|X_A| \geq k + \lceil \log_p k \rceil + 2$ and $|X_B| \geq 1$. For an assignment $\vec{a}_B$, if $f(X_A, \vec{a}_B)$ is totally k-undecomposable, then f has no decomposition of the form $f(X_1, X_2) = g(h_1(X_1), h_2(X_1), \ldots, h_k(X_1), X_2)$, where (X_1, X_2) is a bipartition of X, $|\{X_A\} \cap \{X_1\}| \geq k + 1$, and $|\{X_A\} \cap \{X_2\}| \geq 1 + \lceil \log_p k \rceil$.*

(Proof) Let $X_A = (X_{1A}, X_{2A})$ and $X_B = (X_{1B}, X_{2B})$. Then, apply Lemma 3.1, and we have the theorem. □

Definition 3.2 *Let (X_1, X_2) be a bipartition of $\{x_1, x_2, \ldots, x_n\}$, where $X_1 = (x_1, x_2, \ldots, x_r)$ and $X_2 = (x_{r+1}, x_{r+2}, \ldots, x_n)$. Such a bipartition is compactly denoted by the bipartition of integers $(1, 2, \ldots, r | r + 1, r + 2, \ldots, n)$.*

Example 3.4 *Let $f(x_1, x_2, x_3, x_4, x_5)$ be a five-variable two-valued function. If $f(x_1, x_2, x_3, 0, 0)$ is totally 1-undecomposable, then f is 1-undecomposable for the following 12 bipartitions:*

$(1,2|3,4,5)$, $(1,2,4|3,5)$, $(1,2,5|3,4)$, $(1,2,4,5|3)$,
$(1,3|2,4,5)$, $(1,3,4|2,5)$, $(1,3,5|2,4)$, $(1,3,4,5|2)$,
$(2,3|1,4,5)$, $(2,3,4|1,5)$, $(2,3,5|1,4)$, $(2,3,4,5|1)$.

∎

Theorem 3.2 *Consider a p-valued function $f(X_A, X_B)$, where $n_A = |X_A| \geq k + \lceil \log_p k \rceil + 2$ and $n_B = |X_B| \geq 1$. For an assignment $\vec{a}_B$, if $f(X_A, \vec{a}_B)$ is totally k-undecomposable, then f is k-undecomposable for*

$$\alpha(n_A, n_B, p, k) = \left[\sum_{i=k+1}^{n_A - 1 - \lceil \log_p k \rceil} C(n_A, i) \right] 2^{n_B}$$

bipartitions.

(Proof) F is k-undecomposable when the following conditions are satisfied:

1) More than k variables in $\{X_A\}$ are included as bound variables.
2) More than $\lceil \log_p k \rceil$ variables in $\{X_A\}$ are included as free variables.
3) Variables in $\{X_B\}$ can be either in the bound set or the free set.

From 1) and 2), we have the first factor. And, from 3), we have the second factor. □

Example 3.5 *Let $f(x_1, x_2, x_3, x_4, x_5)$ be a five-variable 2-valued function. If $f(x_1, x_2, x_3, 0, 0)$ is totally 1-undecomposable, then f is 1-undecomposable for $\alpha = 12$ bipartitions, since $k = 1$, $p = 2$, $n_1 = 3$, and $n_2 = 2$. This is also verified by Example 3.4* ∎

Corollary 3.1 *Consider an n-variable function $f(X_A, X_B)$, where $n_A = |X_A| \geq k + \lceil \log_p k \rceil + 2$, and $n_B = |X_B| \geq 1$. For an assignment $\vec{a}_B$, if $(X_A, \vec{a}_B)$ is totally k-undecomposable, then we have to check for at most*

$$\beta(n_A, n_B, p, k) = \left[\sum_{i=0}^{k} C(n_A, i) + \sum_{j=0}^{\lceil \log_p k \rceil} C(n_A, j) \right] 2^{n_B}$$

bipartitions.

(Proof) There are $2^n = 2^{n_A} 2^{n_B}$ bipartitions. Among them, $\alpha(n_A, n_B, p, k)$ bipartitions are k-undecomposable. So, we have to check at most $\beta(n_A, n_B, p, k) = 2^n - \alpha(n_A, n_B, p, k)$ bipartitions. □

Example 3.6 *Corollary 3.1 shows that when $p = 2$ and $k = 1$, the fraction of β to 2^n is $\gamma = \frac{\beta}{2^n} = \frac{n_A + 2}{2^{n_A}}$. Therefore, when $n_A = 3$, $\gamma = 5/8$; when $n_A = 4$, $\gamma = 3/8$; when $n_A = 5$, $\gamma = 7/32$; and when $n_A = 6$, $\gamma = 1/8$.* ∎

IV Switching Function Representing Set of Bipartitions

Functional decomposition is to find a bipartition (X_1, X_2) such that $f(X_1, X_2) = g(h_1(X_1), h_2(X_1), \ldots, h_k(X_1), X_2)$. There are 2^n different bipartitions including trivial ones, and these can be represented by a switching function of n variables. In this part, we will introduce such representations. Also, bipartitions that will not produce decompositions are compactly denoted by symmetric functions. We also introduce notations for symmetric functions.

Definition 4.1 *A function f is a* **totally symmetric function** *if any permutation of the variables in f does not change the function.*

Definition 4.2 *The* **elementary symmetric functions** *of n variables are*

$S_0^n = \bar{x}_1 \bar{x}_2 \cdots \bar{x}_n$,
$S_1^n = x_1 \bar{x}_2 \cdots \bar{x}_n \vee \bar{x}_1 x_2 \bar{x}_3 \cdots \bar{x}_n \vee \cdots \vee \bar{x}_1 \bar{x}_2 \cdots \bar{x}_{n-1} x_n$,
$\cdots\cdots\cdots\cdots\cdots$, *and*
$S_n^n = x_1 x_2 \cdots x_n$.

$S_i^n = 1$ iff exactly i inputs are equal to one. Let $A \subseteq \{0, 1, \ldots, n\}$. A symmetric function S_A^n is defined as follows:

$$S_A^n = \bigvee_{i \in A} S_i^n.$$

Example 4.1 *$f(x_1, x_2, x_3) = x_1 x_2 x_3 \vee x_1 \bar{x}_2 \bar{x}_3 \vee \bar{x}_1 x_2 \bar{x}_3 \vee \bar{x}_1 \bar{x}_2 x_3$ is a totally symmetric function. $f = 1$ when all the variables are one, or when only one variable is one. Thus, f can be written as $S_1^3 \vee S_3^3 = S_{\{1,3\}}^3$.* ∎

Definition 4.3 *A set of bipartitions of the input variables $\{x_1, x_2, \ldots, x_n\}$ is represented by a switching function bp of n variables. In bp, $x_i = 1$ denotes that x_i is in the bound set, and $x_i = 0$ denotes that x_i is in the free set. The number of true minterms of bp is denoted by $|bp|$.*

Example 4.2 *Suppose that $n = 5$. The minterm $x_1 x_2 x_3 \bar{x}_4 \bar{x}_5$ denotes that x_1, x_2, and x_3 are in the bound set, and x_4 and x_5 are in the free set.* ∎

Lemma 4.1 *The set of bipartitions for trivial k-decompositions for n-variable p-valued function is given by*

$$u_0 = S_{\{0,1,\ldots,k\}}^n \vee S_{\{n - \lceil \log_p k \rceil, \ldots, n\}}^n.$$

(Proof) When the number of variables in the bound set is less than $k+1$, then it is a trivial decomposition. To be non-trivial k-decomposition, at least $1 + \lceil \log_p k \rceil$ variables must be in the free set. So, if the number of variables in the bound set is greater than $n - 1 - \lceil \log_p k \rceil$, then it is a trivial decomposition. □

Example 4.3 *Let $n = 10$, $p = 2$, and $k = 2$, then the set of bipartitions for trivial k-decompositions is given by $u_0 = S_{\{0,1,2,9,10\}}^{10}$. This is explained as follows: If the number of bound variables is two or smaller, then*

the decomposition is trivial, since the module for H has two outputs. If the number of variables in the bound set is 9 or 10, then the number of free variables is one or zero. By Definition 2.3, this also corresponds to a trivial decomposition. Thus, the number of trivial decompositions is given by

$$|u_0| = C(10,0) + C(10,1) + C(10,2) + C(10,9) + C(10,10) = 67.$$

The set of non-trivial bipartitions is given by $\bar{u}_0 = S^{10}_{\{3,4,5,6,7,8\}}$. ∎

Theorem 4.1 *Let* $f(X)$ *be a p-valued function, and* (X_A, X_B) *be a partition of* X. *If* $f(X_A, \vec{a}_B) = \hat{f}(x_1, x_2, \ldots, x_r)$ *is totally k-undecomposable, then f has no k-decomposition for the bipartitions*

$$u = S^r_{\{k+1,k+2,\ldots,r-1-\lceil \log_p k \rceil\}}(x_1, x_2, \ldots, x_r).$$

(Proof) By Theorem 3.1 and Lemma 4.1, f is k-undecomposable for these bipartitions. □

Example 4.4 *Consider the case where* $p = 2$, $n = 5$, $k = 1$ *and* $r = 3$. *If* $f(x_1, x_2, x_3, 0, 0)$ *is totally 1-undecomposable, then f is undecomposable for the bipartitions*

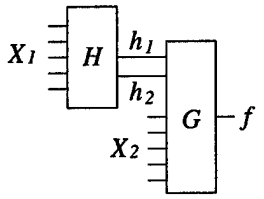
$$u = S^3_2(x_1, x_2, x_3) = \bar{x}_1 x_2 x_3 \vee x_1 \bar{x}_2 x_3 \vee x_1 x_2 \bar{x}_3.$$

Note that u denotes the same set of bipartitions as Example 3.4. ∎

Example 4.5 *Suppose that we have to check whether the given 2-valued 10-variable function* $f(x_1, x_2, \ldots, x_{10})$ *can be realized by a network shown in Fig. 4.1. In this case, the straightforward method needs to check all possible bipartitions* (X_1, X_2), *where* $|X_1| = 5$ *and* $|X_2| = 5$. *This set of bipartitions is represented by* $S^{10}_5(x_1, x_2, \ldots, x_{10})$, *and the number of bipartitions to consider is* $|S^{10}_5(x_1, x_2, \ldots, x_{10})| = C(10,5) = 252$. *However, if* $f(x_1, x_2, x_3, x_4, x_5, 0, 0, 0, 0, 0)$ *is totally 2-undecomposable, then we need not check for* $C(5,3) \times C(5,2) = 100$ *bipartitions. This fact is explained as follows: From Theorem 4.1, the set of bipartitions that will not produce 2-decomposition is given by*

$$u_1 = S^5_3(x_1, x_2, x_3, x_4, x_5) S^5_2(x_6, x_7, x_8, x_9, x_{10}).$$

In u_1, *the first factor selects three variables from* $\{x_1, x_2, x_3, x_4, x_5\}$ *as bound variables, and the second factor selects two variables from* $\{x_6, x_7, x_8, x_9, x_{10}\}$ *as bound variables. For example, suppose that* $\{X_1\} = \{x_1, x_2, x_3, x_6, x_7\}$ *is selected as a bound set, and* $\{X_2\} = \{x_4, x_5, x_8, x_9, x_{10}\}$ *is selected as a free set. This bipartition* (X_1, X_2) *does not produce 2-decomposition, since* $\{x_1, x_2, x_3\}$ *is in the bound set and* $\{x_4, x_5\}$ *is in the free set, and* $f(x_1, x_2, x_3, x_4, x_5, 0, 0, 0, 0, 0)$ *is totally 2-undecomposable. Note that* $|u_1| = C(5,3)C(5,2) = 10 \times 10 = 100$.



$$f(X_1, X_2) = g(h_1(X_1), h_2(X_1), X_2)$$

Figure 4.1: 2-decomposition of 10-variable function.

In a similar way, if $f(0, 0, 0, 0, 0, x_6, x_7, x_8, x_9, x_{10})$ *is also totally 2-undecomposable, then the following bipartitions need not be checked:*

$$u_2 = S^5_3(x_6, x_7, x_8, x_9, x_{10}) S^5_2(x_1, x_2, x_3, x_4, x_5).$$

u_2 *denotes* $C(5,3) \times C(5,2) = 100$ *bipartitions for that no 2-decomposition exist. So, we need only to check for the following bipartitions:*

$$bp = S^{10}_5(x_1, x_2, \ldots, x_{10}) \bar{u}_1 \bar{u}_2.$$

Since u_1 *and* u_2 *are mutually disjoint, we have only to check*

$$|bp| = |S^{10}_5(x_1, x_2, \ldots, x_{10})| - |u_1| - |u_2| = C(10,5) - 100 - 100 = 52$$

bipartitions. In this case, we can reduce the search space into one fifth by finding two subfunctions that are 2-undecomposable. ∎

Theorem 4.2 *Let* $f(x_1, x_2, \ldots, x_{n-1}, a)$ *be totally k-undecomposable, where* $a \in P$. *Then,* $f(x_1, x_2, \ldots, x_{n-1}, x_n)$ *is totally k-undecomposable iff f is undecomposable for the following* $C(n-1, \lceil \log_p k \rceil) + C(n-1, k)$ *bipartitions:* $S^{n-1}_{\{n-1-\lceil \log_p k \rceil\}}(x_1, x_2, \ldots, x_{n-1})\bar{x}_n \vee S^{n-1}_k(x_1, x_2, \ldots, x_{n-1})x_n$.

Example 4.6 *Consider the case where* $n = 6$, $p = 2$, *and* $k = 2$. *Suppose that* $f(x_1, x_2, x_3, x_4, x_5, 0)$ *is totally k-undecomposable. To show that f is totally undecomposable, we need the followings: For* $x_6 = 0$, *we have to check for the bipartitions denoted by* $S^5_4(x_1, x_2, x_3, x_4, x_5)\bar{x}_6$. *Fig. 4.2(a) shows an example of* $C(5,4)$ *bipartitions. For* $x_6 = 1$, *we have to check for bipartitions denoted by* $S^5_2(x_1, x_2, x_3, x_4, x_5)x_6$. *Fig. 4.2(b) shows an example of* $C(5,2) = 10$ *bipartitions.* ∎

(a) 2-decomposition corresponding to $\bar{x}_6$.

(b) 2-decomposition corresponding to x_6.

Figure 4.2:

V Number of Totally k-Undecomposable Functions

When n is sufficiently large, almost all functions are totally 1-undecomposable [19]. In this part, we will show that almost all functions are also totally k-undecomposable.

5.1 p-valued case

Lemma 5.1 *Let $N_d(n,p,k)$ be the number of n-variable p-valued k-decomposable functions. Then,*

$$N_d(n,p,k) \leq \sum_{n_1=k+1}^{n-1-\lceil \log_p k \rceil} C(n,n_1) p^{kp^{n_1}+p^{n-n_1+k}}.$$

(Proof) Suppose that a function f has a disjoint k-decomposition shown in Fig. 1.1. First, consider the module H. The number of ways to select bound variables is $C(n,n_1)$. Since H has n_1 inputs and k outputs, the number of functions for H is $p^{kp^{n_1}}$. Next, consider the module G. Since G has n_2+k inputs and single output, the number of functions for G is $p^{p^{n_2+k}} = p^{p^{n-n_1+k}}$. Hence, we have the lemma. □

Theorem 5.1 *Let $N_{ud}(n,p,k)$ be the number of n-variable p-valued totally k-undecomposable functions. Then,*

$$\frac{N_{ud}(n,p,k)}{p^{p^n}} \to 1 \text{ as } n \to \infty.$$

(Proof) Since $N_{ud}(n,p,k) + N_d(n,p,k) = p^{p^n}$, we will show that

$$\frac{N_d(n,p,k)}{p^{p^n}} \to 0 \text{ as } n \to \infty. \qquad (5.1)$$

From Lemma 5.1, we have

$$N_d(n,p,k) \leq \sum_{n_1=k+1}^{n-1-\lceil \log_p k \rceil} C(n,n_1) p^{kp^{n_1}+p^{n_2+k}}. \qquad (5.2)$$

Since $C(n,n_1) < 2^n$, the right-hand-side of (5.2) is less than

$$2^n \sum_{n_1=k+1}^{n-1-\lceil \log_p k \rceil} p^{kp^{n_1}+p^{n_2+k}} = 2^n \sum_{n_1=k+1}^{n-1-\lceil \log_p k \rceil} p^{A(n_1)}, \qquad (5.3)$$

where $A(n_1) = kp^{n_1}+p^{n_2+k}$. Note that $A(n_1)$ takes its maximum when $n_1 = k+1$ and $n_1 = n-1-\lceil \log_p k \rceil$, and the values of $A(n_1)$ are $kp^{k+1}+p^{n-1}$ and $p^{n-1}+kp^{k+1}$, respectively. Thus, $A(n_1) \leq p^{n-1}+C$, where C does not depend on n. So, (5.3) is less than

$$D(n) = 2^n(n-1-\lceil \log_p k \rceil - (k+1))p^{p^{n-1}+C}.$$

Let us take the logarithm of $D(n)$, and we have

$$\log_p D(n) = n \log_p 2 + \log_p(n-k-\lceil \log_p k \rceil - 2) + p^{n-1} + C.$$

Since $\frac{\log_p D(n)}{p^n} \to \frac{1}{p}$ as $n \to \infty$, we can conclude that (5.1) holds. □

5.2 Two-valued case

When $n \geq 4$ and $p = 2$, most functions are totally 1-undecomposable. For $k = 1$, we obtained the values of $N_{ud}(n,p,k)$ by exhaustive enumeration:

$$N_{ud}(3,2,1) = 104,$$
$$N_{ud}(4,2,1) = 57,240, \text{ and}$$
$$N_{ud}(5,2,1) = 4,290,002,448.$$

When $n = 5$, there are $2^{2^5} = 2^{32} = 4,294,967,296$ functions. Thus, 99.9% of the functions are totally 1-undecomposable. The case of $k = 2$ is interesting, since some FPGAs have LUTs with two outputs [3]. The decompositions must satisfy the relation:

$$|X_1| \geq k+1 \text{ and } |X_2| \geq 1+\lceil \log_2 k \rceil.$$

This requires that $|X| = |X_1|+|X_2| \geq k+2+\lceil \log_2 k \rceil$. Thus, when $k = 2$, only the functions with $n \geq 5$ are interesting. For $n = 5$, the only case is $n_1 = 3$ and $n_2 = 2$, and we have $N_{ud}(5,2,2) = 3,744,402,432$. Thus, 87.2% of the 5-variable functions are totally 2-undecomposable.

VI Conclusion and Comments

In this paper, we defined totally k-undecomposable logic functions, and showed a systematic method to find a set of bipartitions that will not produce disjoint k-decompositions. Key contributions are:

1) Generation of a set of k-undecomposable bipartitions from totally k-undecomposable subfunctions.
2) Representation of k-undecomposable bipartitions by an n-variable switching function.
3) Enumeration of totally k-undecomposable functions.

The presented method can be extended to the case of incompletely specified functions. This method can be combined to existing decomposition methods to reduce search space.

When $n = 3$ or 4, $p = 2$ and $k = 1$, totally k-undecomposable functions are easily detected by BDDs and look-up tables [17]. By using this method, we can show the undecomposability of randomly generated functions very quickly.

We decomposed more than four thousand benchmark functions including functions with 256 inputs and 245 outputs [16, 17]. Experimental results for $p = 2$ and $k = 1$ show that for 1-undecomposable functions, the computation time were reduced to up to one hundreds. Currently, we are developing a system for k-decompositions with $k = 2$.

Even if the given functions have two-valued inputs only, functional decompositions with multi-valued inputs seems to be useful. This is explained as follows: Suppose that a completely specified two-valued input function has a k-decomposition of the form $f(X_1, X_2) = g(h_1(X_1), g_2(X_1), \ldots, g_k(X_1), X_2)$, and that $\mu(f : X_1, X_2) < 2^k$. In this case, assigning $\mu(f : X_1, X_2)$ different binary vectors to the k outputs of H produces don't care conditions for function g. This makes decomposition problem very difficult [21]. However, if we do not assign the binary vectors to the output of h, but assume that h produces a multiple-valued output, then no don't cares are generated. In this case, the decomposition problem is easier.

Acknowledgments

This work was supported in part by a Grant in Aid for Scientific Research of the Ministry of Education, Science, Culture and Sports of Japan. Constructive comments of the reviewers improved the readability.

References

[1] R. L. Ashenhurst, "The decomposition of switching functions," *In Proceedings of an International Symposium on the Theory of Switching*, pp. 74-116, April 1957.

[2] V. Bertacco and M. Damiani, "The disjunctive decomposition of logic functions," *Proc. ICCAD*, pp. 78-82, 1997.

[3] S. D. Brown, R. J. Francis, J. Rose, and Z. G. Vranesic, *Field Programmable Gate Arrays*, Kluwer Academic Publishers, Boston 1992.

[4] S-C. Chang, M. Marek-Sadowska, and T. Hwang, "Technology mapping for LUT FPGA's based on decomposition of binary decision diagrams," *IEEE Trans. CAD*, Vol. CAD-15, No. pp. 1226-1236, Oct. 1996.

[5] H. A. Curtis, *A New Approach to the Design of Switching Circuits*, D. Van Nostrand Co., Princeton, NJ, 1962.

[6] S. Devadas, A. Wang, A. R. Newton, and A. Sangiovanni-Vincentelli, "Boolean decomposition in multilevel logic optimization," *IEEE Journal of Solid-State Circuits*, Vol. 24, pp. 399-408, April 1989.

[7] C. Files, R. Drechsler, and M. Perkowski, "Functional decomposition of MVL functions using multi-valued decision diagrams," *ISMVL*, pp. 27-32, May 1997.

[8] Y-T. Lai, M. Pedram, and S. B. K. Vrudhula, "EVBDD-based algorithm for integer linear programming, spectral transformation, and functional decomposition," *IEEE Trans. CAD*, Vol. 13, No. 8, pp. 959-975, Aug. 1994.

[9] T. Luba, "Decomposition of multiple-valued functions," *International Symposium on Multiple-Valued Logic*, pp. 256-261, Bloomington, Indiana, May 1995.

[10] Y. Matsunaga, "An exact and efficient algorithm for disjunctive decomposition," *SASIMI'98*, pp. 44-50, Oct. 1998.

[11] R. Murgai, R. K. Brayton, and A. Sangiovanni-Vincentelli, *Logic Synthesis for Field-Programmable Gate Arrays*, Kluwer, 1995.

[12] J. P. Roth and R. M. Karp, "Minimization over Boolean graphs," *IBM Journal of Research and Development*, pp. 227-238, April 1962.

[13] T. Sasao, "Application of multiple-valued logic to a serial decomposition of PLA's," *International Symposium on Multiple-Valued Logic*, pp. 264-271, Zangzou, China, May 1989.

[14] T. Sasao, "FPGA design by generalized functional decomposition," in (Sasao ed.) *Logic Synthesis and Optimization*, Kluwer Academic Publishers, 1993.

[15] T. Sasao and J. T. Butler, "On bi-decompositions of logic functions," *ACM/IEEE International Workshop on Logic Synthesis*, Tahoe City, California, May 1997.

[16] T. Sasao and M. Matsuura, "DECOMPOS: An integrated system for functional decomposition," *1998 International Workshop on Logic Synthesis*, Lake Tahoe, June 1998.

[17] T. Sasao, "On a method to accelerate functional decompositions," *Technical Reports of IEICE*, VLD98-58, pp. 103-109, Sept. 1998.

[18] C. Scholl and P. Molitor, "Communication based FPGA synthesis for multi-output Boolean functions," *Asia and South Pacific Design Automation Conference*, pp. 279-287, Aug. 1995.

[19] C. E. Shannon, "The synthesis of two-terminal switching circuits," *Bell Syst. Tech. J.* 28, 1, pp. 59-98, 1949.

[20] V. Y-S, Shen, A. C. Mckellar, and P. Weiner, "A fast algorithm for the disjunctive decomposition of switching functions," *IEEE Trans. Comput.*, Vol. C-20, No. 3, pp. 304-309, March 1971.

[21] W. Wan and M. A. Perkowski, "A new approach to the decomposition of incompletely specified multi-output functions based on graph coloring and local transformations and its application to FPGA mapping," *European Design Automation Conference (Euro DAC)*, pp. 230-235, Sept. 1992.

[22] B. Wurth, K. Eckl, and K. Antreich, "Functional multiple-output decomposition: Theory and implicit algorithm," *Design Automation Conf.*, pp. 54-59, June 1995.

A Generalization of Shestakov's Function Decomposition Method

J. J. Lou
ThoughtWorks Inc.
651 W. Washington Blvd., Suite 500
Chicago, Illinois 60661
USA
jjlou@maveric.uwaterloo.ca

J. A. Brzozowski
Department of Computer Science
University of Waterloo
Waterloo, Ontario
Canada N2L 3G1
brzozo@uwaterloo.ca

Abstract

Shestakov expresses an incompletely specified Boolean function $f(x_1, \ldots, x_n)$ in terms of Boolean functions g_u, g_v and h in the form $h(g_u(u_1, \ldots, u_r), g_v(v_1, \ldots, v_s))$, where $\{u_1, \ldots, u_r\} \cup \{v_1, \ldots, v_s\} = \{x_1, \ldots, x_n\}$. We generalize his method to multi-valued functions with partial don't care's represented in a compact cube-like notation; we do this using blankets, which are generalizations of set systems. Łuba and Selvaraj express a Boolean function $f(x_1, \ldots, x_n)$ in terms of Boolean functions g and h as $h(u_1, \ldots, u_r, g(v_1, \ldots, v_s))$. This method has been formalized using blankets by Brzozowski and Łuba, and generalized to multi-valued functions by Brzozowski and Lou. The relations among these methods are discussed.

1 Introduction

Decomposition of Boolean and multi-valued functions has been studied by many authors since the 1950's; we refer the reader to [2, 7], where various approaches to decomposition are discussed. The present paper is concerned mainly with the method of Shestakov [8, 9], its relation to the techniques of Łuba and Selvaraj [6] and its generalizations [1, 3, 5].

Łuba and Selvaraj developed an approach to Boolean function decomposition using set systems and their generalizations [6]. They express a function $f(x)$ in the form $h(u, g(v))$, where $x = (x_1, \ldots, x_n)$, $u = (u_1, \ldots, u_r)$, $v = (v_1, \ldots, v_s)$, $X = \{x_1, \ldots, x_n\}$, $U = \{u_1, \ldots, u_r\}$, $V = \{v_1, \ldots, v_s\}$, $U \cup V = X$, and g and h are Boolean functions. Very similar methods were discovered independently by Shestakov [8, 9], who expresses a Boolean function $f(x)$ in the form $h(g_u(u), g_v(v))$, where u, v, and x are as above, and g_u, g_v, and h are Boolean functions. The methods of Łuba and Selvaraj [5, 6] have been formalized by Brzozowski and Łuba [2], and extended to the multi-valued case by Brzozowski and Lou [1]. In this paper, we formalize Shestakov's method using blanket algebra [2], and extend it to the multi-valued case.

2 Set Functions and Blankets

A *vector* of distinct variables is denoted by a letter in lower case, and the corresponding *set* of variables, by the same letter in upper case. Thus, if $s = (s_1, \ldots, s_n)$ is a vector of variables, then $S = \{s_1, \ldots, s_n\}$ is the corresponding set. Also, if $S = \{s_1, \ldots, s_n\}$ is a set of variables explicitly represented in that order, then $s = (s_1, \ldots, s_n)$ is the corresponding vector. Suppose $T = \{s_{i_1} \ldots, s_{i_r}\}$ is a subset of S with $i_1 < i_2 < \ldots < i_r$, and $t = (s_{i_1} \ldots, s_{i_r})$ is the corresponding vector. If $b = (b_1, \ldots, b_n)$ is any vector of values, then b^t is the *projection* of b to the variables in t, i.e., $b^t = (b_{i_1}, \ldots, b_{i_r})$. Vectors of values are written without parentheses and commas. For example, suppose $s = (s_1, s_2, s_3, s_4)$, $t = (s_1, s_3, s_4)$ and $b = (4, 1, 2, 0)$; then $b^t = (4, 2, 0)$. In simplified notation $b = 4120$ and $b^t = 420$.

Let E be a finite nonempty set, the set of possible values for each variable, and let $D = 2^E - \{\emptyset\}$. The set D is partially ordered by set inclusion. For $k > 0$, an element of D^k is called a *set vector*. The inclusion partial order is extended to D^k. For $t, t' \in D^k$,

$$t \subseteq t' \text{ if and only if } t_i \subseteq t'_i \text{ for all } i,\ 1 \leq i \leq k.$$

The *intersection* of two elements t and t' from D^k is the component-by-component set intersection. This intersection $t \cap t' = (t_1 \cap t'_1, \ldots, t_k \cap t'_k)$ is empty if one of its components is empty; otherwise, it is an element of D^k.

We use a compact matrix notation to represent multi-valued functions. A $\rho \times (n + m)$ *set matrix* F is a matrix with elements from D. We associate a vector x of input variables with the first n columns of the matrix; also X is

0-7695-0161-3/99 $10.00 © 1999 IEEE

the corresponding set of input variables. Similarly, y and Y are the vector and set of output variables associated with the last m columns of F. Each row t of a set matrix is a set vector of length $n+m$, where the first n components, denoted by t^x, are input values and the last m components, denoted by t^y, are output values. By convention, to simplify matrix F we leave out any row which has $(E,\ldots,E)$ as its output.

An example of a set matrix is given in Table 1, where $E=\{0,1,2\}$, $\rho=8$, $n=5$, and $m=2$. For simplicity, curly brackets have been omitted from the matrix entries.

Table 1. Set matrix F

Row	x_1	x_2	x_3	x_4	x_5	y_1	y_2
1	1	0,1	0	0	0,1	1	0
2	1,2	1	0	1,2	2	0	1,2
3	0,1	0	0	0,1	0	1,2	0
4	2	0,2	1,2	2	2	0,1	2
5	0,1	0	0,2	1	1	2	1,2
6	0	0	0,1	0	2	1	1
7	0,1,2	1	0,2	0	2	0	0
8	0	2	0	0	2	1	2

In the binary case, where $E=\{0,1\}$, a row vector t of F represents a cube, since its entries are $\{0\}$s, $\{1\}$s, and $\{0,1\}$s, representing logic 0s, 1s, and "don't cares," respectively. This is the well-known *fr* format, used in the program ESPRESSO. This notation is also extended to the multi-valued case, where each entry is (a) a singleton set (a particular value), (b) the set E of all possible domain values (the complete don't care), or (c) a nonempty proper subset of E (a partial don't care). Borrowing the terminology from the binary case, we refer to an element $e=(e_1,\ldots,e_n)$ of E^n as a *minterm*. On the other hand, a set vector of the form $b=(\{e_1\},\ldots,\{e_n\})\in D^n$, is a *singleton vector*. The set of all such singleton vectors is denoted by Σ^n. By a slight abuse of terminology, we also refer to singleton vectors as minterms. Furthermore, we refer to the set of rows of a set matrix F also as F, and we write $t\in F$ if t is a row of F. A minterm $b\in\Sigma^n$ is *involved* in a row $t\in F$, if $t^x\supseteq b$. We call b *relevant to* f if b is involved in some row $t\in F$.

A set matrix is said to be *consistent* if, for any set T of rows of F,

$$\bigcap_{t\in T} t^x\neq\emptyset \text{ implies } \bigcap_{t\in T} t^y\neq\emptyset.$$

Consistency means that, if the input projections of the rows in T contain a common minterm b, then the output projections of that set of rows also contain a common minterm. *From now on we consider only consistent set matrices.*

One verifies that the set matrix of Table 1 is consistent. Only Rows 1 and 3 share a common input minterm 10000. These rows share the output minterm 10.

We denote by $F_{x\supseteq b}=\{t\in F\mid t^x\supseteq b\}$ the set of all rows of F involving a given minterm b. For example, for the matrix of Table 1, $F_{x\supseteq 10000}=\{1,3\}$.

The interpretation of a (consistent) set matrix F is as follows. It defines a multi-valued function, called *set function*, $f:\Sigma^n\to D^m$. For any singleton vector $b\in\Sigma^n$,

$$f(b)=\bigcap_{t\in F_{x\supseteq b}} t^y, \text{ if } F_{x\supseteq b}\neq\emptyset$$

and $f(b)=(E,\ldots,E)$ (i.e., m Es—the complete don't care) otherwise. We find it convenient to refer to the vector of variables of f as the *input* of f, and to the vector of function values as the *output* of f.

For example, for the matrix of Table 1, $f(10000)=(\{1\},\{0\})\cap(\{1,2\},\{0\})=(\{1\},\{0\})$, $f(00001)=(\{0,1,2\},\{0,1,2\})$, since 00001 does not appear in any row of F, and $f(11012)=(\{0\},\{1,2\})$, from Row 2.

In the following, we briefly give the definitions and notation used in connection with blankets; for more details see [2]. A *blanket* on a set S is a collection $\beta=\{B_1,\ldots,B_k\}$ of nonempty and distinct subsets of S, called *blocks*, whose union is S. We write $\beta=\{B_i\}$ when it is possible to avoid referring to the number of blocks in β. Also, we write a block with a bar over it to reduce the number of curly brackets. Thus, for example, $\beta=\{\{1,2\},\{2,3\}\}$ is a blanket on the set $\{1,2,3\}$; it will be written $\{\overline{1,2},\overline{2,3}\}$.

The *product* $\beta*\beta'$ of two blankets is defined by

$$\beta*\beta'=ne\{B_i\cap B_j\mid B_i\in\beta \text{ and } B_j\in\beta'\}.$$

where $ne\{S_i\}=\{S_i\}-\{\emptyset\}$ for a set $\{S_i\}$ of subsets of S.

For two blankets β and β' on S, we write $\beta\leq\beta'$ if for each B_i in β there exists a B_j in β' such that $B_i\subseteq B_j$.

Assume the rows in F are numbered $1,\ldots,\rho$. We deal with blankets on the set of rows of F, but refer to them as blankets on the set $\{1,\ldots,\rho\}$, for convenience. Suppose V is an r-element subset of the set $X\cup Y$ of input and output variables. Blanket β_v is defined by

$$\beta_v=ne\{F_{v\supseteq d}\mid d\in\Sigma^r\}.$$

In particular, β_x is the input blanket, corresponding to the set X of input variables, and β_y is the output blanket, corresponding to Y.

For the matrix of Table 1, let $U=\{x_1,x_2,x_3\}$ and $V=\{x_3,x_4,x_5\}$, then

$$\beta_u=\{\overline{3,5,6};\ \overline{6};\ \overline{5};\ \overline{7};\ \overline{8};\ \overline{1,3,5};\ \overline{1,2,7};\ \overline{4};\ \overline{2,7}\},$$

$$\beta_v = \{\overline{1,3};\ \overline{1};\ \overline{6,7,8};\ \overline{3};\ \overline{5};\ \overline{2};\ \overline{6};\ \overline{4};\ \overline{7}\},\ \text{and}$$

$$\beta_y = \{\overline{7};\ \overline{2};\ \overline{2,4};\ \overline{1,3};\ \overline{6};\ \overline{4,8};\ \overline{3};\ \overline{5}\}.$$

3 Double Separation of Set Functions

Let $X = \{x_1, \ldots, x_n\}$ be the set of input variables of a set function f. Let U and V be two subsets of X such that $U \cup V = X$. Without loss of generality, we relabel the variables $x_1, \ldots, x_n$ so that $U = \{x_1, \ldots, x_r\}$ and $V = \{x_{n-s+1}, \ldots, x_n\}$. Thus for a vector x with n components, the vector of the first r components is x^u and the vector of the last s components is x^v. We now define the concept of separation, which is like decomposition, but for which no attention is paid to the number of inputs in the component functions. In a decomposition, the component functions should have fewer inputs than the function being decomposed. Since there is a natural relationship between blankets and separations, we consider separations first. Size constraints can be added separately.

Definition 1 *Let f be a set function specified by set matrix F with $n > 0$ inputs and $m > 0$ outputs, and let (U, V) be as above. Let g_u be a set function with r inputs and p outputs, let g_v be a set function with s inputs and q outputs, and let h be a set function with $p + q$ inputs and m outputs. The triple (g_u, g_v, h) is a* double separation *of f with respect to (U, V), if, for every minterm $b \in \Sigma^n$ relevant to f, we have $g_u(b^u) \in \Sigma^p$, $g_v(b^v) \in \Sigma^q$ and*

$$f(b) \supseteq h(g_u(b^u), g_v(b^v)).$$

To ensure that the inputs to h are well defined, we require that $g_u(b^u)$ and $g_v(b^v)$ be well defined, i.e., that they should be in Σ^p and Σ^q respectively. Note that f and h are not equal, since some don't cares of f can become more specified in h, i.e., become smaller sets of values.

Suppose we are given a consistent set matrix $F(x)$ with n inputs, a set $V \subseteq X$ of cardinality s, and a set function $g(v)$ with s inputs and p outputs. Furthermore, assume that $g(b^v) \in \Sigma^p$ for all minterms $b \in \Sigma^n$ that are relevant to F. Let $e \in \Sigma^p$, and let

$$F_{w=e} = \{t \in F \mid \exists d \in \Sigma^s : t^v \supseteq d \text{ and } g(d) = e\}$$

be the set of all rows of F that can produce the value e for output vector w. Note that $F_{w=e}$ is always a union of blocks of the form $F_{v \supseteq d}$. Blanket γ_g is defined by

$$\gamma_g = ne\{F_{w=e}\}.$$

Note that γ_g is uniquely determined by F, V, and g.

Theorem 1 *Let $f(x)$ be a set function specified by a set matrix F, and let (U, V) be a pair of subsets of X satisfying $U \cup V = X$. For every pair of blankets β_{g_u} and β_{g_v} satisfying the conditions*

- $\beta_u \leq \beta_{g_u}$, $\beta_v \leq \beta_{g_v}$, *and*
- $\beta_{g_u} * \beta_{g_v} \leq \beta_y$,

there is a double separation (g_u, g_v, h) of f such that

$$\gamma_{g_u} \leq \beta_{g_u} \text{ and } \gamma_{g_v} \leq \beta_{g_v}.$$

Intuitively, the first condition states that the information present in the inputs in vector u should be sufficient to compute the outputs of function g_u. Similarly, the second condition states that vector v should be sufficient to compute the outputs of function g_v. The third condition requires that the information contained in the outputs of g_u and g_v should be sufficient to reproduce output y. The conditions $\gamma_{g_i} \leq \beta_{g_i}$ can be interpreted as stating that function g_i *corresponds* to blanket β_{g_i}. Often, $\gamma_{g_i} = \beta_{g_i}$, but the inequality permits us to use more blankets.

The separation in [2] is a special case of the separation described here. The proof of Theorem 1 is constructive [4], and similar to the corresponding proof in [2]. The converse of Theorem 1 is also true, provided U and V are disjoint; the proof is similar to the one in [2]. Thus the counterexample for the converse of the theorem in that paper also serves as a counterexample here, when U and V are not disjoint.

We illustrate the construction required in the proof of Theorem 1 using the function of Table 1, $u = (x_1, x_2, x_3)$, $v = (x_3, x_4, x_5)$, and blankets

$$\beta_{g_u} = \{\overset{0}{\overline{1,3,5,6}};\ \overset{1}{\overline{1,2,7}};\ \overset{2}{\overline{4,8}}\},\ \text{and}$$

$$\beta_{g_v} = \{\overset{0}{\overline{1,3,4}};\ \overset{1}{\overline{6,7,8}};\ \overset{2}{\overline{2,5}}\}.$$

We can check that these blankets satisfy the conditions of Theorem 1. Each blanket has only three blocks; thus we can code the blocks of each blanket by a single ternary variable, whose value is shown above each block. Function g_u is shown in Table 2 in a somewhat compacted form to save space. Consider $(x_1, x_2, x_3) = 000$. Minterm 000 appears in rows 3, 5, and 6 of F, i.e., it corresponds to block $\{3, 5, 6\}$ of β_u. Since $\beta_u \leq \beta_{g_u}$, each block of β_u is contained in at least one block of β_{g_u}. In this case $\{3, 5, 6\} \subseteq \{1, 3, 5, 6\}$. Since $\{1, 3, 5, 6\}$ has been assigned the value 0, we set $g_u(000) = 0$. The remaining entries are computed similarly.

Function g_v is computed in the same way; it is shown in Table 3.

To construct function $h = h(w^u, w^v)$, we first find block B of β_{g_u} corresponding to input w^u and block B' of β_{g_v}

Table 2. Set matrix defining g_u

x_1	x_2	x_3	w^u
0	0	0,1,2	0
0	1	0,2	1
0	2	0	2
1	0	0,2	0
1	1	0,2	1
2	0	1,2	2
2	1	0,2	1
2	2	1,2	2

corresponding to input w^v. Next we calculate the intersection $\hat{B} = B \cap B'$. The output is then the intersection of the outputs of the rows contained in $\hat{B}$. If $B \cap B'$ is nonempty then it is contained in a block of β_y, since $\beta_{g_u} * \beta_{g_v} \leq \beta_y$. Thus, the intersection of the output vectors is also nonempty. For example, for $w^u w^v = 00$, the block of β_{g_u} is $B = \overline{1,3,5,6}$ and the block of β_{g_v} is $B' = \overline{1,3,4}$. The intersection of the two blocks is $\hat{B} = \overline{1,3}$. Hence the output is $(\{1\},\{0\}) \cap (\{1,2\},\{0\}) = (\{1\},\{0\})$. The rest of the table for h follows similarly.

Table 3. Set matrix defining g_v

x_3	x_4	x_5	w^v
0	0	0,1	0
0	0	2	1
0	1	0	0
0	1	1,2	2
0	2	2	2
1,2	0	2	1
1,2	2	2	0
2	1	1	2

The table of h for our example is shown in Table 4.

Table 4. Set matrix defining h

w^u	w^v	z_1	z_2
0	0	1	0
0	1	1	1
0	2	2	1,2
1	0	1	0
1	1	0	0
1	2	0	1,2
2	0	0,1	2
2	1	1	2

The form $h(u_1,\ldots,u_r,g(v_1,\ldots,v_s))$ described in [2] is called a *single separation*. It is a special case of the separation above, where function g_u is the identity function. Double separation is equivalent to two single separations performed in sequence. The verification of the following proposition is routine [4].

Proposition 1 *Let f be a set function and let (g_u, g_v, h) be a double separation of f. Let r, q and m be as before. Then there exists a set function h' with $r+q$ inputs and m outputs such that (g_v, h') is a single separation of f and (g_u, h) is a single separation of h'.*

The reader can verify Proposition 1 for our running example by constructing a function h' for a single separation.

4 Table Reduction Method

We now describe Shestakov's method [9] for finding blankets β_{g_u} and β_{g_v}. The method can be used for both single and double separations. We first derive a single separation and then a double one. Note that our single separation is of the form $h(g_u(u), v)$, so the usual roles of U and V are reversed.

For simplicity, we use a binary function as an example. Let f be a Boolean function with five inputs and two outputs specified by the matrix in Table 5. Note that Φ is a short hand for $\{0,1\}$ and curly brackets are omitted for all other entries. This example is taken from [9].

Table 5. Matrix defining f.

Row	x_1	x_2	x_3	x_4	x_5	y_1	y_2
1	0	0	1	0	1	0	1
2	0	1	1	Φ	Φ	1	1
3	1	Φ	0	0	1	Φ	0
4	Φ	1	Φ	1	0	Φ	1
5	1	1	1	0	Φ	0	1
6	0	Φ	0	Φ	1	0	0
7	0	0	1	1	1	1	0
8	1	Φ	0	1	1	Φ	1

Let $U = \{x_1, x_2, x_3\}$ and $V = \{x_3, x_4, x_5\}$. We calculate β_u and β_v and label their blocks by the integers corresponding to the binary input combinations. Hence, for example, block B_5 is missing from β_u as $(x_1, x_2, x_3) = 101$ is not present in the matrix defining f.

$$\beta_u = \{\overset{B_0}{\overline{6}}; \overset{B_1}{\overline{1,7}}; \overset{B_2}{\overline{4,6}}; \overset{B_3}{\overline{2,4}}; \overset{B_4}{\overline{3,8}}; \overset{B_6}{\overline{3,4,8}}; \overset{B_7}{\overline{4,5}}\},$$

$$\beta_v = \{\overset{B'_1}{\overline{3,6}}; \overset{B'_2}{\overline{4}}; \overset{B'_3}{\overline{6,8}}; \overset{B'_4}{\overline{2,5}}; \overset{B'_5}{\overline{1,2,5}}; \overset{B'_6}{\overline{2,4}}; \overset{B'_7}{\overline{2,7}}\}.$$

We construct a *block multiplication table* where rows are labeled B_i and columns B'_j. To define the entries in the table, we need the following definitions.

Definition 2 *Let W be a subset of X with k elements. Let B be a block in blanket β_w. Then $b \in \Sigma^k$ is a* characteristic minterm *of B if B is the set of all rows of F that involve b, i.e., if $B = F_{w \supseteq b}$.*

For example, if $W = U$ for the function of Table 5, then 100 is a characteristic minterm of B_4, but 110 is not, since $F_{u \supseteq 100} = \{3, 8\} = B_4$ and $F_{u \supseteq 110} = \{3, 4, 8\} = B_6$.

Definition 3 *Given a block B_i of β_u and a block B'_j of β_v, we define $B_i \wedge B'_j = B_i \cap B'_j$ if $B_i \cap B'_j \neq \emptyset$ and there exists a minterm b relevant to f such that b^u is a characteristic minterm of B_i and b^v is a characteristic minterm of B'_j. Otherwise, $B_i \wedge B'_j = \emptyset$.*

Intuitively, the operation $\wedge$ is intersection if there is a common minterm which defines both blocks, and is empty otherwise. In fact, we can show that in the case $U \cap V = \emptyset$, this operation is equivalent to block intersection. Entry $T_{i,j}$ in table T is now defined to be $B_i \wedge B'_j$. The block multiplication table for our function is presented in Table 6. For clarity, empty sets are represented by $-$ and curly brackets are omitted for all other entries.

Table 6. Block multiplication table

	B'_1	B'_2	B'_3	B'_4	B'_5	B'_6	B'_7
B_0	6	–	6	–	–	–	–
B_1	–	–	–	–	1	–	7
B_2	6	4	6	–	–	–	–
B_3	–	–	–	2	2	2,4	2
B_4	3	–	8	–	–	–	–
B_6	3	4	8	–	–	–	–
B_7	–	–	–	5	5	4	–

Note that $T_{2,6}$ is empty instead of $B_2 \cap B'_6 = \{4\}$. This is because all characteristic minterms of B_2 satisfy $x_3 = 0$, while all characteristic minterms of B'_6 satisfy $x_3 = 1$; thus minterm b of our definition does not exist, and $B_2 \wedge B'_6 = \emptyset$. Now we construct the *output table* by replacing $T_{i,j}$ by $\bigcap_{t \in T_{i,j}} t^y$ if $T_{i,j}$ is not empty, and by the complete don't care, otherwise. We use $-$ to denote the complete don't care. The output table for our example is shown in Table 7.

Definition 4 *A subset $S = \{B_{i_1}, \ldots, B_{i_k}\}$ of blocks of β_u is called* compatible *if, in each column of the output table, the entries in rows $i_1, \ldots, i_k$ have a nonempty intersection.*

We now find a blanket β_{B_i} on the set β_u; the blocks of β_{B_i} are compatible sets of blocks of β_u. We obtain a blanket β on F by replacing each block with the union of its

Table 7. Output table

	B'_1	B'_2	B'_3	B'_4	B'_5	B'_6	B'_7
B_0	00	–	00	–	–	–	–
B_1	–	–	–	–	01	–	10
B_2	00	Φ1	00	–	–	–	–
B_3	–	–	–	11	11	11	11
B_4	Φ0	–	Φ1	–	–	–	–
B_6	Φ0	Φ1	Φ1	–	–	–	–
B_7	–	–	–	01	01	Φ1	–

elements. We then use blanket β as β_{g_u} to obtain a single separation. Note that this is equivalent to Łuba and Selvaraj's method [6], where blocks are merged in β_u to get β such that $\beta_v * \beta \leq \beta_y$. For some graph theoretical methods for finding β see [7].

In our example, one possible blanket β_{B_i} on the set β_u is $\beta_{B_i} = \{\overline{B_0, B_1, B_2, B_7}; \overline{B_3, B_4, B_6}\}$. The corresponding blanket β_{g_u} with one possible encoding is

$$\beta_{g_u} = \{\overline{1, 4, 5, 6, 7}^{0}; \overline{2, 3, 4, 8}^{1}\}.$$

We can check that $\beta_{g_u} * \beta_v \leq \beta_y$, since

$$\beta_y = \{\overline{3, 6}; \overline{1, 4, 5, 7}; \overline{3, 7}; \overline{2, 4, 8}\}.$$

Thus β_{g_u} corresponds to a single separation of f.

Denote the block of β_{g_u} encoded 0 by δ_0 and the block encoded 1 by δ_1. We construct the *row-merged output table* as follows. The rows of the table are labeled δ_l and the columns, B'_j. The entry (l, j) of the table is the intersection of entries (i_k, j) in the row compatibility table where $\delta_l = \{B_{i_k}\}$. This table is shown in Table 8.

Table 8. Row-merged output table

	B'_1	B'_2	B'_3	B'_4	B'_5	B'_6	B'_7
δ_0	00	Φ1	00	01	01	Φ1	10
δ_1	Φ0	Φ1	Φ1	11	11	11	11

Now we use a similar procedure to find compatible columns. From our table, we obtain the following blanket on the set β_v: $\beta_{B'_j} = \{\overline{B'_1}; \overline{B'_2, B'_4, B'_5, B'_6}; \overline{B'_3}; \overline{B'_7}\}$. The corresponding blanket β_{g_v} with one possible encoding is $\beta_{g_v} = \{\overline{3, 6}^{00}; \overline{1, 2, 4, 5}^{01}; \overline{6, 8}^{10}; \overline{2, 7}^{11}\}$.

We label the four blocks $\delta'_0, \ldots, \delta'_3$. After merging columns we obtain Table 9. We can get function h directly from this table: each input combination of h is formed by concatenating the encodings of δ_i and δ'_j, while the output vector is the entry (i, j) of the table.

Table 9. Final Table

	δ'_0	δ'_1	δ'_2	δ'_3
δ_0	00	01	00	10
δ_1	$\Phi 0$	11	$\Phi 1$	11

Note that we could have merged columns before rows in Table 7. This is equivalent to interchanging sets U and V. If we do this, we get $\beta_{B'_j} = \{\overline{B'_1, B'_4, B'_5, B'_6}; \overline{B'_2, B'_7}; \overline{B'_3}\}$, which results in $\beta_{g_v} = \{\overline{1,2,3,4,5,6}; \overline{2,4,7}; \overline{6,8}\}$.

We then find $\beta_{B_i} = \{\overline{B_0, B_2}; \overline{B_1, B_7}; \overline{B_3}; \overline{B_4, B_6}\}$, and $\beta_{g_u} = \{\overline{4,6}; \overline{1,4,5,7}; \overline{2,4}; \overline{3,4,8}\}$.

Table 10. New Final Table

	δ'_0	δ'_1	δ'_2
δ_0	00	$\Phi 1$	00
δ_1	01	10	–
δ_2	11	11	–
δ_3	$\Phi 0$	$\Phi 1$	$\Phi 1$

The new final table is shown in Table 10. Note that this is a 4×3 table as opposed to the 2×4 table we obtained earlier. Thus the order in which we reduce our table is important. In our example, the set of tables for our second double separation would have larger total size than the tables for our first double separation.

Theorem 2 *The table reduction method described above results in a double separation of the original function f.*

Proof. Let b be a minterm relevant to f. Then b^u is a characteristic minterm of a block B_i in blanket β_u and b^v is a characteristic minterm of a block B'_j in blanket β_v. We know that $B_i \cap B'_j \neq \emptyset$, as the row that contains b is in the intersection. By definition, we have $B_i \wedge B'_j = B_i \cap B'_j$. In our final table obtained using the procedure, there is an entry corresponding to the value $h(g_u(b^u), g_v(b^v))$. This entry is the intersection of some output vectors, one of which is the entry $T_{i,j}$ in the output table. Since that entry is simply $f(b)$, we get $f(b) \supseteq h(g_u(b^u), g_v(b^v))$ as required.

5 Conclusions

We have formalized Shestakov's decomposition method using blanket algebra, we have generalized it to multi-valued functions represented by cubes, we have related it to the work of Łuba and Selvaraj, and we have proved its correctness.

We close by mentioning some related issues. One problem that we have not discussed so far is the selection of the sets U and V for a decomposition. This can be done by trial and error; one can also use certain necessary conditions on blankets β_u and β_y to reduce the search. We refer the reader to [2] for more details.

Further theoretical issues related to blanket algebra and its application to function decomposition are discussed in [1, 2]. Blanket methods for decomposition have been implemented by Nowicka and Łuba in a package called DEMAIN. Implementation issues, experimental results and comparisons with other decomposition methods are discussed in [2, 7].

Acknowledgment: This research was supported by the Natural Sciences and Engineering Research Council of Canada under a postgraduate scholarship and grant No. OGP0000871. This work was done while the first author was at the University of Waterloo.

Note: References [1, 2, 3, 4, 7] can be obtained from the web at `http://maveric.uwaterloo.ca/publication.html`

References

[1] J. A. Brzozowski and J. J. Lou, Blanket Algebra for Multiple-Valued Function Decomposition, *Algebraic Engineering, Proc. Workshop on Formal Languages and Comp. Systems,* Kyoto, Japan, 24–28 March 1997, C. L. Nehaniv and M. Ito, eds., World Scientific, 1998.

[2] J. A. Brzozowski and T. Łuba, *Decomposition of Boolean Functions Specified by Cubes, Part I: Theory of Serial Decompositions Using Blankets,* Research Report CS–97–01, Dept. of Comp. Sci., University of Waterloo, Waterloo, ON, Canada, Jan. 1997; REVISED October 1998.

[3] J. J. Lou, *Decompositions of Multi-Valued Functions,* Master of Math. Thesis, Dept. of Comp. Sci., University of Waterloo, Waterloo, ON, Canada, August 1998.

[4] J. J. Lou and J. A. Brzozowski, *A Formalization of Shestakov's Decomposition,* Research Report CS-98-03, Dept. of Comp. Sci., University of Waterloo, Waterloo, ON, Canada, February 1998.

[5] T. Łuba, Decomposition of Multiple-Valued Functions, *Proc. 25th International Symposium on Multiple-Valued Logic,* Bloomington, Indiana, pp. 256–261, May 1995.

[6] T. Łuba and H. Selvaraj, A General Approach to Boolean Function Decomposition and its Applications in FPGA-Based Synthesis, *VLSI Design,* Vol. 3, Nos. 3–4, pp. 289–300, 1995.

[7] M. Nowicka and T. Łuba, *Decomposition of Boolean Functions Specified by Cubes, Part II: Practical Results,* Research Report CS–97–01 REVISED, Dept. of Comp. Sci., University of Waterloo, Waterloo, ON, Canada, October 1998.

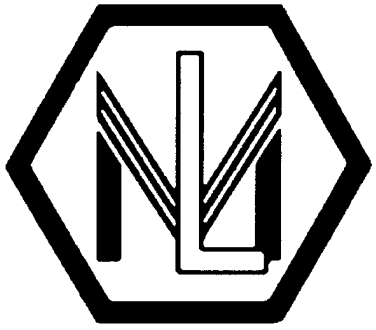
[8] E. Shestakov, Decomposition of Systems of Completely Defined Boolean Functions by Argument Covering, *Automatic Control and Comp. Sci.,* Vol. 28, No. 1, pp. 12–20, 1994.

[9] E. Shestakov, Decomposition of Systems of Incompletely Defined Boolean Functions by Argument Cover, *Automatic Control and Comp. Sci.,* Vol. 28, No. 6, pp. 4–15, 1994.

SESSION IIIB

CLONES

CHAIR: Grant Pogosyan



Gigantic Pairs of Minimal Clones

Ivo G. Rosenberg* and Hajime Machida†

* Département de mathématiques et de statistique,
Université de Montréal, Montréal, Québec H3C 3J7 Canada

† Department of Mathematics,
Hitotsubashi University, Kunitachi, Tokyo 186-8601 Japan

Abstract

In 1992, L. Szabó asked for the minimal number $n = n(|A|)$ such that the clone of all operations on A can be generated as the join of n minimal clones. He showed, e.g., *$n(p) = 2$ for any prime p, and later G. Czédli proved in 1998 that if k has a divisor ≥ 5 then $n(k) = 2$.*

In this paper, a pair (f, g) of operations is called gigantic if each of f and g generates a minimal clone and the set $\{f, g\}$ generates the clone of all operations. First, we give a general theorem to characterize a gigantic pair. Then we show that $n(k) = 2$ for every k which is not a power of 2.

1 Introduction

Let A be a finite set with $k(> 1)$ elements. Let $\mathcal{O}_A^{(n)}$ be the set of all n-ary operations from A^n into A and let $\mathcal{O}_A = \bigcup_{n=1}^{\infty} \mathcal{O}_A^{(n)}$. Denote by $\mathcal{J}_A$ be the set of all projections pr_i^n $(1 \leq i \leq n)$ over A where pr_i^n is defined as $pr_i^n(x_1, \ldots, x_i, \ldots, x_n) = x_i$ for every $(x_1, \ldots, x_n)$ in A^n. Often, the index A is omitted from $\mathcal{O}_A^{(n)}$ or $\mathcal{O}_A$.

A subset C of $\mathcal{O}_A$ is a **clone** on A if (i) C contains $\mathcal{J}_A$ and (ii) C is closed under (functional) composition. The set of all clones on A is a lattice with respect to the inclusion relation. It is called the lattice of clones on A and is denoted by $\mathcal{L}_A$.

In the lattice $\mathcal{L}_A$ an atom of $\mathcal{L}_A$ is called a minimal clone. In other words, a clone C on A is a **minimal clone** if (i) $C \neq \mathcal{J}_A$ and (ii) $\mathcal{J}_A \subset C' \subseteq C$ implies $C' = C$ for any clone C' on A. Dually, a coatom of $\mathcal{L}_A$ is called a maximal clone.

In contrast to maximal clones which have been completely determined in terms of relations ([Ro 65], [Ro 70A]), minimal clones are not yet fully known in spite of some extensive efforts. The known facts include, among others, the complete list of minimal clones on a three-element set ([Cs 83]) and the type classification of minimal clones on arbitrary A ([Ro86]). For other informations on minimal clones, refer to [Pá 86], [Qu 95], [Szc 96], etc. In relation to minimal clones, the authors [MR 93] studied essentially minimal clones which, by definition, sit at the bottom of $\mathcal{L}_A$ excluding clones generated by unary operations.

L. Szabó raised the question in [Sza 92]: Determine the minimal number $n = n(k)$ such that the clone of all operations on A can be generated as the join of n minimal clones. He asserted that $2 \leq n(k) \leq 3$ for every $k > 1$ and showed $n(p) = 2$ for any prime p. Later, G. Czédli [Cz 98] proved that if k has a divisor ≥ 5 then $n(k) = 2$.

In this paper, after defining the term gigantic for a pair (f, g) of operations in $\mathcal{O}_A$ (Definition 1.2), we give a theorem to characterize a gigantic pair (Section 2) and prove the existence of a gigantic pair for every k that is not a power of 2 by explicitly constructing an example of such pair (Section 3).

Now it is obvious that a minimal clone can be generated by a single operation.

Definition 1.1 *An operation f on A is* **minimal** *if* (i) *it generates a minimal clone and* (ii) *every operation from $[f]$ whose arity is smaller than the arity of f is a projection. (Here, $[f]$ is the clone generated by f.)*

0-7695-0161-3/99 $10.00 © 1999 IEEE

The types of minimal operations are known in [Ro 86]. An operation $f \in \mathcal{O}$ is **idempotent** if $f(x,\dots,x) \approx x$. (Here, $f(x,\dots,x) \approx x$ means $f(x,\dots,x) = x$ for all $x \in A$.)

Theorem 1.1 *([Ro 86]) Every minimal operation is of one of the following five types:*

1) Unary operations $f \in \mathcal{O}^{(1)}$ such that either (i) $f^2 (= f \circ f) = f$ or (ii) f is a permutation of prime order p.

2) Idempotent binary operations; i.e., *$f \in \mathcal{O}^{(2)}$ such that $f(x,x) \approx x$.*

3) Majority operations; i.e., *$f \in \mathcal{O}^{(3)}$ such that $f(x,x,y) \approx f(x,y,x) \approx f(y,x,x) \approx x$.*

4) Semiprojections (or quasiprojections); i.e., *$f \in \mathcal{O}^{(n)}$ $(3 \leq n \leq k)$ such that there exists $i\,(1 \leq i \leq n)$ satisfying $f(a_1,\dots,a_n) = a_i$ whenever $a_1,\dots,a_n \in A$ are not pairwise distinct.*

5) If $k = 2^m$, the ternary operations $f(x,y,z) \approx x + y + z$ where $< A\ ; + >$ is an elementary 2-group (i.e., the additive group of an m-dimensional vector space over GF(2)).

Inspired by [Cz 98], we give the following definition.

Definition 1.2 *A pair (f,g) of minimal operations is called* **gigantic** *if $\{f,g\}$ is complete. (i.e., $[f,g] = \mathcal{O}$ or, equivalently, $< A\ ; f,g >$ is primal.)*

Before we proceed, we shall review some terminology from universal algebra.

Definition 1.3 **A** $= < A; F >$ *is an* **algebra** *if A is a set and F is a set of finitary operations defined over and taking values in A. When $F = \{f_1,\dots,f_t\}$,* **A** *is also expressed as $< A; f_1,\dots,f_t >$.*

Let **A** $= < A; F >$ *be an algebra.*

A subset B of A is a **subuniverse** *of* **A** *if B is closed under every operation f in F. A subuniverse B of $< A; F >$ is a* **proper** *subuniverse if $\phi \subset B \subset A$.*

An equivalence relation θ on A is a **congruence** *of* **A** *if θ satisfies the property that for every operation f in F, if f is n-ary, $x_1 \theta y_1, \dots, x_n \theta y_n$ implies $f(x_1,\dots,x_n)\,\theta\, f(y_1,\dots,y_n)$ for all $x_1,\dots,x_n,y_1,\dots,y_n \in A$. A congruence θ of* **A** *is* **proper** *if θ is not a trivial equivalence relation. An algebra* **A** *is* **simple** *if it has no proper congruences.*

Moreover, a permutation φ on A is an **automorphism** *of* **A** *if φ satisfies the property that for every operation f in F, if f is n-ary,*

$$f(\varphi(x_1),\dots,\varphi(x_n)) = \varphi(f(x_1,\dots,x_n))$$

for all $x_1,\dots,x_n \in A$. The set of all automorphisms of **A** *is denoted by Aut* **A**. *An automorphism φ is* **proper** *if φ is not the identity permutation id_A of A.*

Finally in this section, we supply some definitions concerning relations.

Definition 1.4 *For a set A, an h-***ary relation** *on A is a subset of the Cartesian product A^h. For an n-ary operation f in $\mathcal{O}_A$ and an h-ary relation ρ on A, f is said to* **preserve** *ρ if $(x_{1j}, x_{2j}, \dots, x_{hj}) \in \rho$ for every $j = 1,2,\dots,n$ implies*

$$(f(x_{11},x_{12},\dots,x_{1n}),\dots,f(x_{h1},\dots,x_{hn})) \in \rho\,.$$

Definition 1.5 *Let $k = h^m$ where $h > 2$ and $m \geq 1$. A set $T = \{\theta_1,\dots,\theta_m\}$ of equivalence relations on k is* **regular**, *or T is a* **regular system** *on k, if* (i) *each block (equivalence class) of every θ_i has h elements and* (ii) *if B_i is a block of θ_i for all $i = 1,\dots,m$ then*

$$|\, B_1 \cap \dots \cap B_m \,| \geq 1\,.$$

Next, the relation determined by T is the relation λ_T defined as the set of all $(a_1,\dots,a_h) \in k^h$ such that for each $i = 1,\dots,m$ it holds that $a_r \theta_i a_s$ for some $1 \leq r < s \leq h$.

2 Characterization Theorem

In this section we characterize a gigantic pair and for this purpose we need the following definitions. Hereafter A is assumed to be the set $k = \{0,1,\dots,k-1\}$ $(k > 1)$.

Definition 2.1 *For a divisor p of k denote by F_p the set of all permutations of k with $\ell := k/p$ cycles of length p.*

Let $f \in F_p$ have cycles $C_0,\dots,C_{\ell-1}$.

An equivalence relation θ on k is **transversal** *to f if there exist* 1) *an equivalence relation λ on ℓ distinct from the least equivalence relation on ℓ and* 2) *an element $c_i \in C_i$ for each $i \in \ell$ such that*

$$\theta = \{(f^m(c_i), f^m(c_j)) \mid i\,\lambda\, j,\ 0 \leq m \leq p-1\}\,.$$

A permutation ψ of k is **orthogonal** *to f if* 1) *$\psi \in F_q$ for some prime divisor q of k,* 2) *$f \circ \psi = \psi \circ f$ and* 3) *if $q \neq p$ then each cycle of ψ meets every C_i in at most a singleton.*

Now, we shall characterize gigantic pairs.

Theorem 2.1 *Let $f \in \mathcal{O}^{(m)}$ and $g \in \mathcal{O}^{(n)}$ where $m \leq n$ and let $\mathbf{A} := < \boldsymbol{k}; g >$. Then the pair (f,g) is gigantic if and only if the following conditions (i) – (vii) are satisfied.*

(i) *$m = 1$ and $f \in F_p$ for some prime divisor p of k (with cycles $C_0, \ldots, C_{\ell-1}$).*

(ii) *$n > 1$ and g is minimal.*

(iii) *$C_{i_1} \cup \cdots \cup C_{i_h}$ is not a proper subuniverse of $\mathbf{A}$ for any $0 \leq i_1 < \cdots < i_h \leq \ell - 1$.*

(iv) *No congruence of $\mathbf{A}$ is transversal to f.*

(v) *No automorphism of $\mathbf{A}$ is orthogonal to f.*

(vi) *Let $k = h^m$ where $h > 2$ and $m > 1$. If there exist:* a) *a permutation φ of m of order 1 or p and permutations $f_0, \ldots, f_{m-1}$ of h such that*

α) $f_{d_0} f_{d_1} \cdots f_{d_{p-1}} = id_{\boldsymbol{h}}$ for each cycle $(d_0 \ldots d_{p-1})$ of φ,

β) for every fixed point i of φ the permutation f_i is of order 1 or p and

γ) for some fixed point j of φ the permutation f_j is fixed-point-free and of order p and b) *a bijection*

$$\psi : x \longmapsto \widehat{x} = (x^{(0)}, \ldots, x^{(m-1)})$$

of $\boldsymbol{k}$ onto $\boldsymbol{h}^m$ such that for all $x \in \boldsymbol{k}$

$$\widehat{f(x)} = (f_0(x)^{(\varphi(0))}, \ldots, f_{m-1}(x)^{(\varphi(m-1))}), \quad (1)$$

then g does not preserve the h-ary relation

$$\{(a_1, \ldots, a_h) \in \boldsymbol{k}^h \mid \#\{a_1^{(i)}, \ldots, a_h^{(i)}\} \leq h-1 \text{ for every } i = 0, \ldots, m-1\}.$$

(vii) a) *Let $k = p^m$ for a prime p and $m \geq 1$,* b) *let $x \longmapsto \widehat{x}$ be a bijection from $\boldsymbol{k}$ onto $\boldsymbol{p}^m$ (the latter considered as the set of all $m \times 1$ matrices over $\boldsymbol{p}$) and* c) *let for all $x \in \boldsymbol{k}$*

$$\widehat{f(x)} = A\widehat{x} + B \quad (2)$$

where $A = P^{-1}JP$, $B = P^{-1}B'$ with P a nonsingular (over GF(p)) $m \times m$ matrix over $\boldsymbol{p}$, J the $m \times m$ Jordan matrix with diagonal $(1, \ldots, 1)$ and blocks of sizes $\ell_1, \ldots, \ell_h$ and $B' = (b_1, \ldots, b_m) \in \boldsymbol{p}^m$ is such that

$$(I + J + J^2 + \cdots + J^{p-1})B' = O_{m \times 1}, \quad (3)$$

and $b_{\ell_1 + \cdots + \ell_u} \neq 0$ for some $1 \leq u \leq h$. Then g does not preserve the quaternary relation

$$\{(x,y,z,t) \in \boldsymbol{k}^4 \mid \widehat{t} = \widehat{x} \ominus \widehat{y} \oplus \widehat{z}\}. \quad (4)$$

Proof ($\Rightarrow$) Let the pair (f,g) be gigantic. As f and g are minimal, they are of the five types 1) – 5) listed in the previous section. Notice that every minimal operation h of type 2) – 5) is idempotent and so $\{0\}$ is a subuniverse of $< \boldsymbol{k}; h >$. Since $\{f,g\}$ is complete, at least one of f and g is of the first type. As $m \leq n$, clearly f is of the first type and, consequently, is unary. Now g cannot be unary due to the completeness of $\{f,g\}$. Thus $n > 1$ and g is idempotent. We have proved (ii). To prove (i), suppose to the contrary that $f^2 = f$. Choose a in the image of f. Thus $f(a) = a$ and, since $g(a, \ldots, a) = a$, $\{a\}$ is a subuniverse of $\mathbf{B} = < \boldsymbol{k}; f, g >$ which contradicts to the completeness of $\{f,g\}$. It follows that $f^p(x) \approx x$ for some prime p. Suppose that f has a fixed point $a \in \boldsymbol{k}$. Then, again, $\{a\}$ is a subuniverse of $\mathbf{B}$. Thus f is fixed-point-free and $f \in F_p$, proving (i). Each of the conditions (iii) – (vii) stipulates that $\{f,g\}$ does not belong to certain maximal clone.

($\Leftarrow$) Let $\{f,g\}$ satisfy the conditions (i) – (vii). Clearly the permutation f is surjective. By (ii), g is essentially more than unary and, being idempotent, it is also surjective.

For a set B of surjective operations containing an essentially more than unary operation, the general completeness criterion from [Ro 65, Ro 70A] was simplified in [Ro 70B] (Theorem 3) to the following criterion:

For a set B with the above properties, the set B is complete if and only if
(A) $< \boldsymbol{k}; B >$ is simple and has no proper subuniverse and no proper automorphism,
(B) If $k = p^m$ for a prime p and $m \geq 1$ and if $< \boldsymbol{k}; + >$ is an elementary abelian p-group then some $b \in B$ does not preserve the quaternary relation

$$\{(x,y,z,x-y+z) \mid x,y,z \in \boldsymbol{k}\},$$

and
(C) If $k = h^m$ with $h > 2$ and $m > 1$ and T is an h-regular system on $\boldsymbol{k}$ then some $b \in B$ does not preserve λ_T.

In view of this criterion, it suffices to verify the conditions (A), (B), and (C) for $\mathbf{B} = < \boldsymbol{k}; f, g >$.

Here we shall only consider the condition (A). Let $\mathbf{C} := < \boldsymbol{k}; f >$ and consider a subuniverse S of $\mathbf{C}$. If a cycle C_i of f meets S then clearly $C_i \subseteq S$; thus the subuniverses of $\mathbf{C}$ are exactly the sets $C_{i_1} \cup \cdots \cup C_{i_h}$ with $0 \leq i_1 < \ldots < i_h \leq \ell - 1$. Now the condition (iii) guarantees that $\mathbf{B} = < \boldsymbol{k}; f, g >$ has no proper subuniverse.

Next we assume to the contrary that there exists a proper congruence θ of $\mathbf{B}= < \boldsymbol{k} ; f, g >$. Consider a block B of θ.

Claim 1 B is either of the form

$$C_{i_1} \cup \cdots \cup C_{i_h} \tag{5}$$

for some $0 \leq i_1 < \ldots < i_h < \ell$, $h \neq \ell$, or $|B \cap C_i| \leq 1$ for all $0 \leq i < \ell$.

Proof. Suppose $|B \cap C_i| > 1$ for some $0 \leq i < \ell$. Then there exist $a \in C_i$ and $0 < m < p$ such that $a \theta f^m(a)$. As f preserves θ, clearly $f^{mi}(a) \theta f^{m(i+1)}(a)$ for all $i = 0, \ldots, p-1$ (where $f^0(a) = a$). Now p being a prime, the set $\{m0, m1, \ldots, m(p-1)\}$ coincides modulo p with p and therefore $C_i \subseteq B$. Now suppose that $|B \cap C_j| = 1$ for some $0 \leq j < \ell$, $j \neq i$. Then $B \cap C_j = \{b\}$ for some $b \in \boldsymbol{k}$. Choose $a \in C_i$. Clearly $a \theta b$ implies $f(a) \theta f(b)$ where $f(a) \in C_i$ and $f(b) \in C_j$. Now $a \theta f(a)$, $f(a) \theta f(b)$ and $a \in C_i \subseteq B$ show $f(b) \in B \cap C_j = \{b\}$, which is a contradiction to f being fixed-point-free. This implies that B is of the form (5) and the claim follows.

Claim 2 $|B \cap C_i| \leq 1$ for all $0 \leq i < \ell$.

Proof. Suppose to the contrary that B is of the form (5). Since the operation g also preserves θ, it maps B^n into a block B' of θ. However, g is idempotent, and so, for every $b \in B$ obviously $b = g(b, \ldots, b) \in B'$, proving $B' = B$. Thus g maps B^n into B and so B is a proper subuniverse of $\mathbf{B} = < \boldsymbol{k} ; f, g >$. This is against the assumption (iii), and proves the claim.

From Claim 2 it is easy to see that θ is transversal to f, in contradiction to (iv). Thus $\mathbf{B}$ is indeed simple.

Now suppose to the contrary that there exists a proper automorphism φ of $\mathbf{B}$.

Claim 3 Each proper automorphism φ of $\mathbf{B}$ is fixed-point-free.

Proof. It is easy to see that $F := \{x \in \boldsymbol{k} \mid \varphi(x) = x\}$ is a subuniverse of $\mathbf{B}$. We have shown above that $\mathbf{B}$ has no proper subuniverse. As $\varphi \neq id_{\boldsymbol{k}}$, clearly $F \subset \boldsymbol{k}$ and so $F = \phi$, proving the claim.

Claim 4 If φ is a proper automorphism of $\mathbf{B}$ then $\varphi \in F_r$ for some proper divisor r of k.

Proof. Suppose to the contrary that φ has a cycle Γ of length c and a cycle of length d where $c < d$. Then φ^c is a proper automorphism of $\mathbf{B}$ whose fixed points include Γ. This contradicts Claim 3 and proves the claim.

Claim 5 $F_q \cap Aut\,\mathbf{B}$ is nonvoid for some prime divisor q of k.

Proof. By Claim 4, $\varphi \in F_r$ for some proper divisor r of k. Choose a prime divisor q of r and set $u := r/q$. Then $\psi := \varphi^u$ is a proper automorphism of $\mathbf{B}$ and $\psi \in F_q$, proving the claim.

Claim 6 There exists an automorphism ψ of $\mathbf{B}$ which is orthogonal to f.

Proof. By Claim 5, there exists $\psi \in F_q \cap Aut\,\mathbf{B}$. Clearly ψ satisfies the conditions 1) and 2) in the definition of orthogonality. Suppose that $\psi(a) = f^m(a)$ for some $a \in \boldsymbol{k}$ and $0 < m < p$. Then $\{f^{m0}(a), \ldots, f^{m(p-1)}(a)\}$ is the cycle C_i containing a and hence $q = p$. This proves the claim.

We have shown that ψ is orthogonal to f. However, this contradicts (v).

Thus we have verified that the condition (A) holds for $\mathbf{B}$. Due to the lack of space we shall omit the proof for the conditions (B) and (C). □

The conditions (i) – (vii) in Theorem 2.1 simplify if k is a prime. We state the result without proof.

Corollary 2.1 *Let k be a prime, and let $f \in \mathcal{O}^{(m)}$ and $g \in \mathcal{O}^{(n)}$ where $m \leq n$. Let $\mathbf{A} := < \boldsymbol{k} ; g >$. Then the pair (f, g) is gigantic if and only if*

(i) *$m = 1$ and f is a cyclic permutation of $\boldsymbol{k}$,*

(ii) *$n > 1$ and g is minimal,*

(v') *f is not an automorphism of $\mathbf{A}$, and*

(vii') *If $n = 2$, $0 < b < k$ and $\psi : x \longmapsto \widehat{x}$ is a permutation of $\boldsymbol{k}$ such that for all $x \in \boldsymbol{k}$ $\widehat{f(x)} = x \oplus b$ then for no $a \in \boldsymbol{k}$ the operation*

$$g'(x, y) :\approx \psi(g(\psi^{-1}(x), \psi^{-1}(y)))$$

is of the form $ax \oplus (1 - a)y$.

Example 1. Let $k = 2$ (Boolean case). There are 4 nonunary minimal operations, namely, $\vee$, $\wedge$, d and r, where $\vee$ and $\wedge$ are the disjunction (OR) and the conjunction (AND), $d(x, y, z) \approx (x \wedge y) \vee (y \wedge z) \vee (z \wedge x)$ and $r(x, y, z) \approx x \oplus y \oplus z$. Notice that $\neg x$ is an automorphism of d and that r is quasi-affine. The remaining two pairs $(\neg, \vee)$ and $(\neg, \wedge)$ both satisfy the conditions of Corollary 2.1. This proves that for $k = 2$ there are exactly 2 gigantic pairs.

Corollary 2.2 [Sz 92] *A gigantic pair exists for every prime k.*

Proof In Corollary 2.1, set $f(x) :\approx x \oplus 1$ and also set $g(x,y)$ for all $x,y \in \boldsymbol{k}$ such that $g(x,y) = x$ if $x = y$ and $g(x,y) = k-1$ if $x \neq y$. It is easy to see that g is minimal; in fact, g is a semilattice join whose order is $0 \leq k-1, \ldots, k-2 \leq k-1$. Every automorphism of $< \boldsymbol{k}; g >$ fixes the element $k-1$, and this proves (v'). To see (vii'), it suffices to note that an operation $ax \oplus (1-a)y$ takes each value from $\boldsymbol{k}$ exactly k times which is not true for g and hence also not true for g'. □

3 The existence of a gigantic pair for k not a power of 2

In this section, we show that a gigantic pair exists for any k that is not a power of 2.

Theorem 3.1 *If k is not a power of 2 then there exists a gigantic pair.*

Proof Let p denote the greatest prime divisor of k and let $\ell := k/p$. Clearly $p > 2$ and in view of Corollary 2.2 we may assume that $\ell > 1$.

First, we define a unary operation f and a binary operation g on $\boldsymbol{k}$ and, then, prove that the pair of those operations is in fact a gigantic pair.

We define f by setting

$$f(ip+j) := \begin{cases} ip+j+1 & \text{if } i \in \boldsymbol{\ell},\ 0 \leq j < p-1, \\ ip & \text{if } i \in \boldsymbol{\ell},\ j = p-1. \end{cases}$$

Thus the cycle C_i of f is $(ip, ip+1, \ldots, ip+p-1)$ for all $i \in \boldsymbol{\ell}$; *i.e.*, f is expressed in the cyclic notation as

$$f = (0\,1 \cdots p-1)\,(p\,p+1 \cdots 2p-1) \cdots \\ \cdots ((\ell-1)p\ (\ell-1)p+1 \cdots k-1).$$

An algebra $\mathbf{A} = < \boldsymbol{k}; \vee >$ is a semilattice if the binary operator $\vee$ is associative, commutative and idempotent (*i.e.*, it satisfies $x \vee (y \vee z) = (x \vee y) \vee z$, $x \vee y = y \vee x$ and $x \vee x = x$ for all $x,y \in \boldsymbol{k}$). The binary relation $\leq$ on $\boldsymbol{k}$ corresponding to $\mathbf{A}$ is defined by setting $a \leq b$ whenever $a \vee b = b$. It is well-known that $\leq$ is a (partial) order in which $a \vee b$ is the join of a and b. Conversely, an order $\leq$ on $\boldsymbol{k}$ in which each pair has a join determines a semilattice.

Let g be the following semilattice operator $\vee$ on $\boldsymbol{k}$. For every x, y in $\boldsymbol{k}$ set

$$x \vee y = \begin{cases} x & \text{if } x = y, \\ (i+1)p & \text{if } x = ip+j,\ y = ip+j' \\ & \quad (0 \leq i \leq \ell-2, \\ & \quad 1 \leq j, j' \leq p-1,\ j \neq j'), \\ 0 & \text{otherwise}. \end{cases}$$

The Hasse diagram of the order corresponding to the semilattice $< \boldsymbol{k}; \vee >$ is in Figure 1. Notice that it is a tree.

We verify the conditions (i) – (vii) of Theorem 2.1.

I) The condition (i) clearly holds.

II) It is well-known that a semilattice join is a minimal operation; indeed, it generates exactly one essentially n-ary operation, namely, $x_1 \vee \cdots \vee x_n$ and from it $x \vee y$ is obtained as $x \vee y \vee \cdots \vee y$.

III) Let $0 \leq i_1 < \ldots < i_h \leq \ell - 1$ be such that $C = C_{i_1} \cup \cdots \cup C_{i_h}$ is a subuniverse of $\mathbf{A} = < \boldsymbol{k}; \vee >$. Since $p \geq 3$, it is clear that $i_j p + 1,\ i_j p + 2 \in C_{i_j}$ for each $1 \leq j \leq h$. Consequently, $(i_j + 1)p = (i_j p + 1) \vee (i_j p + 2)$ belongs to C and therefore $i_{j+1} = i_j + 1$. It follows that $\{i_1, \ldots, i_h\} = \{i_1, i_1+1, \ldots, \ell-1\}$. By the same reason we have $0 \in C$, proving $i_1 = 0$. Thus $C = \boldsymbol{k}$ and (iii) holds.

IV) First we remark that each block B of a congruence θ of a semilattice $\mathbf{A}$ is a subsemilattice of $\mathbf{A}$. Now, suppose to the contrary that there exists a congruence θ of the semilattice $\mathbf{A}$ transversal to f. Let λ be the corresponding equivalence on $\boldsymbol{\ell}$ and let $\{i_0, \ldots, i_{h-1}\}$ be a nonsingleton block of λ (where $0 \leq i_0 < \ldots < i_{h-1} \leq \ell - 1$). We distinguish two cases. A) Suppose $i_0 > 0$. Consider the block B of θ containing $i_0 p$. Clearly $B \cap C_{i_1} = \{i_1 p + c_1\}$. Here $(i_0 p) \vee (i_1 p + c_1) = 0 \notin B$ due to $i_0 > 0$ which contradicts to the above remark. B) Suppose $i_0 = 0$. Consider the block B of θ containing 1 and let $B \cap C_{i_1} = \{i_1 p + c_1\}$. If $c_1 > 0$ then, again, $1 \vee (i_1 p + c_1) = 0 \notin B$ and we are done. Thus let $c_1 = 0$. Consider the block B' of θ containing 2. Then, by the definition, $B' \cap C_{i_1} = \{i_1 p + 1\}$ and $2 \vee (i_1 p + 1) = 0 \notin B'$. This contradiction proves the claim.

V) It is obvious that every semilattice automorphism fixes its greatest element. Since every permutation of $\boldsymbol{k}$ orthogonal to f is fixed-point-free, it is clear that no automorphism of $\mathbf{A}$ is orthogonal to f.

Here again we omit the proof for VI) and VII) because of the lack of space.

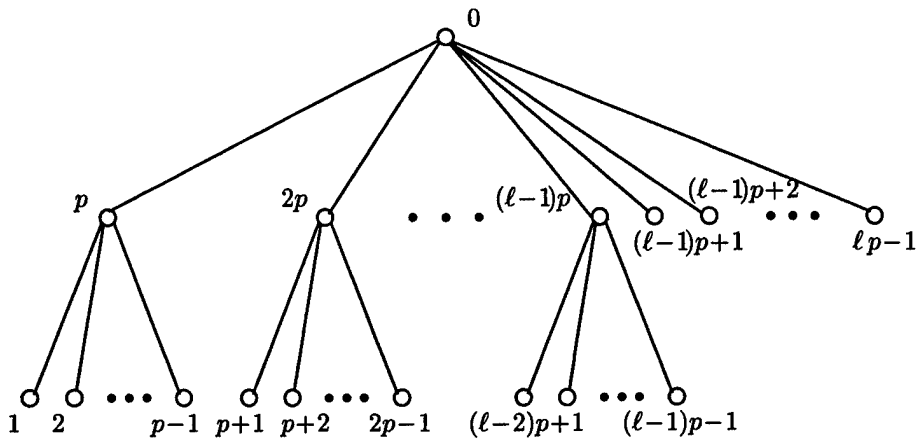


Figure 1: The semilattice for the operation $\vee$

Thus, the pair $(f, \vee)$ is proved to satisfy (i) – (vii) and so it is gigantic by Theorem 2.1. □

4 Conclusion

We defined a "gigantic pair" as a pair of minimal operations which together generate the clone of all operations and then gave a necessary and sufficient condition for a pair of operations to be gigantic.

Next, we constructed a gigantic pair for each k that is not a power of 2. This is a generalization of the results of Szabo and Czédli.Thus, we now know that a gigantic pair exists for every k with $k \neq 2^m$ $(m \geq 2)$.

The remaining case $k = 2^m$ $(m \geq 2)$ has to be considered separately. Notice that the minimal operation g can be neither a majority operation nor a quasiprojection, because if it is one of them then each two-element subset of $\boldsymbol{k}$ is a subuniverse of $< \boldsymbol{k}; g >$ which contradicts to (iii). For simplicity, identify $\boldsymbol{k}$ with $\boldsymbol{2}^m$ and set $f(x) \approx x \oplus e$ (where $\oplus$ is the mod 2 componentwise sum on $\boldsymbol{2}^m$ and $e := (1, \ldots, 1)$). Then the operation $g(x, y, z) :\approx x \oplus y \oplus z$ does not satisfy (vii) and so g is not of this type. Thus, if a gigantic pair (f, g) exists then g must be binary and idempotent. Unfortunately, due to (iii) the operation g cannot be a semilattice join.

NOTE: After the completion of this paper, the authors were informed that the existence of gigantic pairs for every $k > 1$, including k being a power of 2, was established by G. Czédli, R. Halaš, K.A. Kearnes, P.P. Pálfy and Á. Szendrei[CHKPS]. Their work and ours were carried out independently and in parallel.

References

[Cz 98] Czédli, G., Two minimal clones whose join is gigantic, Preprint, 1998.

[Cs 83] Csákány, B., All minimal clones on a three-element set, *Acta Cybernet.*, 6, 1983, 227-238.

[MR 93] Machida, H. and Rosenberg, I.G., Essentially minimal groupoids, in *Algebras and Orders*, Kluwer Academic Publishers, 1993, 287-316.

[Pá 86] Pálfy, P.P., The arity of minimal clones, *Acta Sci. Math. (Szeged)*, 50, 1986, 331-333.

[Qu 95] Quackenbush, R.W., A survey of minimal clones, *Aequat. Math.*, 50, 1995, 3-16.

[Ro 65] Rosenberg, I.G., La structure des fonctions de plusieurs variables sur un ensemble fini, *Comptes rendus de l'Acad. sci. Paris*, 260, 1965, 3817-3819.

[Ro 70A] Rosenberg, I.G., Über die funktionale Vollständigkeit in dem mehrwertigen Logiken, *Rozpravy Čs. Akademie Věd. Ser. Math. Nat. Sci. Praha*, 80, 4, 1970, 3-93.

[Ro 70B] Rosenberg, I.G., Complete sets for finite algebras, *Math. Nachr.*, 44, 1970, 253-258.

[Ro 86] Rosenberg, I.G., Minimal clones I: The five types, *Colloq. Math. Soc. J. Bolyai*, 43, 1986, 405-427.

[Sza 92] Szabó, L., On minimal and maximal clones, *Acta Cybernetica*, 10, 1992, 323-327.

[Sza 97] Szabó, L., On minimal and maximal clones II, *Acta Cybernetica*, Submitted.

[Szc 96] Szczepara, B., Minimal clones generated by groupoids, Ph. D. Thesis (Université de Montréal), 1996.

[CHKPS] Czédli, G., Halaš, R., Kearnes, K.A., Pálfy, P.P. and Szendrei, Á., The join of two minimal clones and the meet of two maximal clones, Preprint.

Maximal chains of partial clones containing all idempotent partial functions

Lucien Haddad and Jean Fugère *

Abstract

Let $k \geq 2$ and $\mathbf{k}$ be a k-element set. We show that every partial clone containing all idempotent partial functions on $\mathbf{k}$ is finitely generated. Moreover we construct maximal chains of length 2^k of such partial clones.

1 Preliminaries

Let $k \geq 2$ and $\mathbf{k} := \{0, \ldots, k-1\}$. For a positive integer n, an n-ary partial function on $\mathbf{k}$ is a map f : dom $(f) \to \mathbf{k}$ where dom (f) is a subset of $\mathbf{k}^n$ called the *domain* of f. Denote by $\mathrm{Par}^{(n)}(\mathbf{k})$ the set of all n-ary partial functions on $\mathbf{k}$ and let $\mathrm{Par}(\mathbf{k}) := \bigcup_{n \in \mathbf{N}} \mathrm{Par}^{(n)}(\mathbf{k})$.

A partial function $g \in \mathrm{Par}^{(n)}(\mathbf{k})$ is a *subfunction* of $f \in \mathrm{Par}^{(n)}(\mathbf{k})$ (in symbols $g \leq f$) if dom $(g) \subseteq$ dom (f) and $g(\underset{\sim}{a}) = f(\underset{\sim}{a})$ for all $\underset{\sim}{a} \in$ dom (g). Moreover let $\mathrm{Op}^{(n)}(\mathbf{k}) := \{f \in \mathrm{Par}^{(n)}(\mathbf{k}) \mid \text{dom } (f) = \mathbf{k}^n\}$ and $\mathrm{Op}(\mathbf{k})$ be the set of all total (or everywhere defined) functions on $\mathbf{k}$, i.e.,

$$\mathrm{Op}(\mathbf{k}) := \bigcup_{n \in \mathbf{N}} \mathrm{Op}^{(n)}(\mathbf{k}) .$$

In the sequel we will say function for total function. We define the superposition of partial functions as follows. For $n, m \geq 1$, $f \in \mathrm{Par}^{(n)}(\mathbf{k})$ and $g_1, \ldots, g_n \in \mathrm{Par}^{(m)}(\mathbf{k})$, the *superposition* of f and $g_1, \ldots, g_n$, denoted $f[g_1, \ldots, g_n]$, is the m-ary partial function h on $\mathbf{k}$ defined by

$$\text{dom } (h) := \{\underset{\sim}{a} \in \mathbf{k}^m \mid \underset{\sim}{a} \in \bigcap_{i=1}^{n} \text{dom } (g_i) \text{ and } (g_1(\underset{\sim}{a}), \ldots, g_n(\underset{\sim}{a})) \in \text{dom } (f)\};$$

and

$$h(\underset{\sim}{a}) := f(g_1(\underset{\sim}{a}), \ldots, g_n(\underset{\sim}{a}))$$

for all $\underset{\sim}{a} \in$ dom (h).

*Département de mathématiques et informatique, collège militaire royal du Canada, B. P. 17000, STN Forces, Kingston, ON, K7K 7B4 Canada. The first author is financially supported by NSERC Canada Operating Grants and ARP Grants.

For every positive integer n and every $1 \leq i \leq n$, let e_i^n denote the *n-ary i-th projection* defined by $e_i^n(a_1, \ldots, a_n) = a_i$ for all $(a_1, \ldots, a_n) \in \mathbf{k}^n$, (notice that e_i^n is a total function). Furthermore let

$$\mathbf{J}(\mathbf{k}) := \{e_i^n \mid 1 \leq i \leq n < \infty\}$$

be the set of all projections on $\mathbf{k}$.

Definition 1. A *partial clone* on $\mathbf{k}$ is a composition closed subset of $\mathrm{Par}(\mathbf{k})$ containing the set of all projections $\mathbf{J}(\mathbf{k})$. A partial clone C contained in the set of all functions $\mathrm{Op}(\mathbf{k})$ is called *a clone* on $\mathbf{k}$. A partial clone C is *strong* if it contains all subfunctions of its functions; i.e., if for every $f \in C$ and $g \in \mathrm{Par}(\mathbf{k})$, $g \leq f \Longrightarrow g \in C$. Following [2], we define the *strong closure* Str (C) of the partial clone C as the smallest strong partial clone containing C. Note that

$$\text{Str } (C) := \{g \in \mathrm{Par}(\mathbf{k}) \mid g \leq f \text{ for some } f \in C\}.$$

Clearly a partial clone C is strong if and only if $C =$ Str (C). It is shown in [2] that a partial clone on $\mathbf{k}$ is strong if and only if it contains the set Str $(\mathbf{J}(\mathbf{k}))$ of all partial projections on $\mathbf{k}$.

Let $h \geq 1$, let ρ be an h-ary relation on $\mathbf{k}$ and let f be an n-ary partial function on $\mathbf{k}$. We say that f *preserves* ρ if for every $h \times n$ matrix $M = [M_{ij}]$ whose columns $M_{*j} \in \rho$, $(j = 1, \ldots n)$ and whose rows $M_{i*} \in$ dom (f) $(i = 1, \ldots, h)$, the h-tuple $(f(M_{1*}), \ldots, f(M_{h*})) \in \rho$. Set $\mathrm{pPol}\,(\rho) := \{f \in \mathrm{Par}(\mathbf{k}) \mid f \text{ preserves } \rho\}$ and $\mathrm{Pol}\,\rho := \mathrm{pPol}\,\rho \cap \mathrm{Op}(\mathbf{k})$.

Let $n \geq 1$ and $f \in \mathrm{Par}^{(n)}(\mathbf{k})$. We say that f is *idempotent* if for all $x \in \mathbf{k}$,

$$(x, \ldots, x) \in \text{dom } (f) \Rightarrow f(x, \ldots, x) = x.$$

Notice that $\mathcal{I} := \bigcap_{x \in \mathbf{k}} \mathrm{pPol}\,\{x\}$ is the set of all idempotent partial functions and $\bigcap_{x \in \mathbf{k}} \mathrm{Pol}\,\{x\}$ is the set of all idempotent functions on $\mathbf{k}$. We refer the reader to [4] for examples and more details.

In this paper we carry on the study, initiated in [4], of the partial clones containing all partial idempotent functions on $\mathbf{k}$. We show that every partial clone that

0-7695-0161-3/99 $10.00 © 1999 IEEE

contain the set of all partial idempotent functions is finitely generated. Moreover, we construct families of maximal chains of such partial clones of size 2^k.

The counterpart for clones of the result we want to establish for partial clones is known. We will use the following result due to R. Quackenbush.

Proposition 1 ([6]) *Let $k \geq 2$. Then the clone $\bigcap_{x \in \mathbf{k}} \mathrm{Pol}\,\{x\}$ of all idempotent functions on $\mathbf{k}$ is generated by all binary idempotent functions.* ∎

The concept of "separating clone" defined in [2] and [3] will be used to show one of our main results.

Definition A clone C on $\mathbf{k}$ is *separating* if there exists $m \geq 1$ such that for all $n \geq 1$ and all $\underset{\sim}{b} \in \mathbf{k}^n$, there exist $g_1, \ldots, g_m \in C \cap \mathrm{Op}^{(n)}(\mathbf{k})$ such that for every $\underset{\sim}{a} \in \mathbf{k}^n$,
$(g_1(\underset{\sim}{a}), \ldots, g_m(\underset{\sim}{a})) = (g_1(\underset{\sim}{b}), \ldots, g_m(\underset{\sim}{b})) \iff \underset{\sim}{a} = \underset{\sim}{b}$.
We employ the following result from [2] (see also [3]).

Proposition 2 *Let $k \geq 2$ and C be a clone on $\mathbf{k}$. Then the partial clone $\mathrm{Str}\,(C)$ is finitely generated if and only if C is a finitely generated separating clone.* ∎

A description of all partial clones containing all idempotent partial functions is given in [4]. We will use the following notation introduced in [4]. For $A := \{a_1, \ldots, a_m\} \subseteq \mathbf{k}$ with $a_1 < \ldots < a_m$, denote by $\mathcal{P}_A$ the partial clone determined by the m-ary relation $\{(a_1, \ldots, a_m)\}$, i.e.,
$\mathcal{P}_A := \mathrm{pPol}\,\{(a_1, \ldots, a_m)\} = \bigcup_{n \in \mathbb{N}} \{f \in \mathrm{Par}^{(n)}(\mathbf{k}) \mid [(a_i, \ldots, a_i) \in \mathrm{dom}\,(f)\ \forall\, i = 1, \ldots, m] \Rightarrow f(a_i, \ldots, a_i) = a_i,\ \forall\, i = 1, \ldots, m\}$.
For instance if $\mathcal{F} = \{\{0\}, \{0,1,2\}\}$, then
$$\bigcap_{X \in \mathcal{F}} \mathcal{P}_X = \mathrm{pPol}\,\{0\} \cap \mathrm{pPol}\,\{(0,1,2)\}.$$
We have

Proposition 3 ([4]) *Let $k \geq 2$. There are finitely many partial clones on $\mathbf{k}$ that contain the set $\mathcal{I}$ of all idempotent partial functions on $\mathbf{k}$. Furthermore, a partial clone C on $\mathbf{k}$ contains $\mathcal{I}$ if and only if there is a family $\mathcal{F}$ of subsets of $\mathbf{k}$ such that $C = \bigcap_{X \in \mathcal{F}} \mathcal{P}_X$.* ∎

Notice that $\mathrm{Par}(\mathbf{k}) = \bigcap_{x \in \emptyset} \mathrm{pPol}\,\{x\}$.

2 New results

2.1 In this first subsection we show that each of the finitely many partial clones on $\mathbf{k}$ that contain $\mathcal{I}$ is finitely generated. We start with

Lemma 4 *The partial clone $\mathcal{I}$ is finitely generated.*

Proof. We show that $\bigcap_{x \in \mathbf{k}} \mathrm{Pol}\,\{x\}$ satisfies the conditions in Proposition 2 and moreover that $\mathcal{I} = \mathrm{Str}\,(\bigcap_{x \in \mathbf{k}} \mathrm{Pol}\,\{x\})$. As mentioned in the section above, the fact that $\bigcap_{x \in \mathbf{k}} \mathrm{Pol}\,\{x\}$ is a finitely generated clone is a consequence of [6]. One may also prove this result by using the Baker-Pixley criterion which states that every clone that contain a near-unanimity function is finitely generated (see [1] and [3]). Now it is easy to see that $\bigcap_{x \in \mathbf{k}} \mathrm{Pol}\,\{x\}$ contains the ternary near-unanimity function g defined by $g(x_1, x_2, x_3) = y$ if at least two of x_1, x_2, x_3 are equal to y and $g(x_1, x_2, x_3) = 0$ if $x_1 \neq x_2 \neq x_3 \neq x_1$. Next we show that $\bigcap_{x \in \mathbf{k}} \mathrm{Pol}\,\{x\}$ is a separating clone on $\mathbf{k}$. Choose $m = 2$ and let $n \geq 1$ and $\underset{\sim}{b} \in \mathbf{k}^n$. Denote by $\omega^{(n)}$ the n-ary diagonal relation on $\mathbf{k}$ defined by $\omega^{(n)} := \{(x, \ldots, x) \mid x \in \mathbf{k}\}$. We distinguish two cases.

Case 1: $\underset{\sim}{b} \in \omega^{(n)}$. Then $\underset{\sim}{b} = (b, \ldots, b)$ for some $b \in \mathbf{k}$. Choose $c \in \mathbf{k} \setminus \{b\}$ and define the n-ary function

$$g(\underset{\sim}{x}) = \begin{cases} e_1^n(\underset{\sim}{x}), & \text{if } \underset{\sim}{x} \in \omega^{(n)} \\ c, & \text{otherwise,} \end{cases}$$

note that $g^{-1}(b) = \{(b, \ldots, b)\}$, and consequently $(g(\underset{\sim}{a}), g(\underset{\sim}{a})) = (g(\underset{\sim}{b}), g(\underset{\sim}{b})) \iff \underset{\sim}{a} = \underset{\sim}{b},\ \forall \underset{\sim}{a} \in \mathbf{k}^n$.
Case 2: $\underset{\sim}{b} \notin \omega^{(n)}$. Define the two n-ary functions g_1 and g_2 by

$$g_1(\underset{\sim}{x}) = \begin{cases} e_1^n(\underset{\sim}{x}), & \text{if } \underset{\sim}{x} \in \omega^{(n)} \\ 0, & \text{if } \underset{\sim}{x} = \underset{\sim}{b}, \\ 1, & \text{otherwise,} \end{cases}$$

and

$$g_2(\underset{\sim}{x}) = \begin{cases} e_1^n(\underset{\sim}{x}), & \text{if } \underset{\sim}{x} \in \omega^{(n)} \\ 1, & \text{if } \underset{\sim}{x} = \underset{\sim}{b}, \\ 0, & \text{otherwise,} \end{cases}$$

Thus $(g_1(\underset{\sim}{b}), g_2(\underset{\sim}{b})) = (0,1)$. Let $\underset{\sim}{a} \in \mathbf{k}^n$, $\underset{\sim}{a} \neq \underset{\sim}{b}$. Then $g_1(\underset{\sim}{a}) = g_1(\underset{\sim}{b}) \Rightarrow \underset{\sim}{a} = (0, \ldots, 0)$ and so $g_2(\underset{\sim}{a}) = 0 \neq g_2(\underset{\sim}{b}) = 1$.

Finally we show that $\mathcal{I} = \mathrm{Str}\,(\bigcap_{x\in\mathbf{k}} \mathrm{Pol}\,\{x\})$. Let $f \in \mathcal{I} \setminus \bigcap_{x\in\mathbf{k}} \mathrm{Pol}\,\{x\}$ be n-ary and define the function g by setting

$$g(\underset{\sim}{x}) = \begin{cases} f(\underset{\sim}{x}), & \text{if } \underset{\sim}{x} \in \mathrm{dom}\,(f) \\ e_1^n(\underset{\sim}{x}), & \text{if } \underset{\sim}{x} \in \omega^{(n)} \setminus \mathrm{dom}\,(f) \\ 0, & \text{otherwise.} \end{cases}$$

Let $\underset{\sim}{x} = (x_1, \ldots, x_n) \in \omega^{(n)}$. If $\underset{\sim}{x} \in \mathrm{dom}\,(f)$, then $g(\underset{\sim}{x}) = f(\underset{\sim}{x}) = x_1$ since $f \in \mathcal{I}$. Else if $\underset{\sim}{x} \notin \mathrm{dom}\,(f)$, then $g(\underset{\sim}{x}) = x_1$ by definition, thus $g(\underset{\sim}{x}) = x_1$ for all $\underset{\sim}{x} \in \omega^{(n)}$, i.e., $g \in \bigcap_{x\in\mathbf{k}} \mathrm{Pol}\,\{x\}$ and by definition g is an extension of the function f. ∎

Combining the above lemma with Proposition 3 we get

Corollary 5 *Let $k \geq 2$. Every partial clone that contains the set $\mathcal{I}$ of all partial idempotent functions is finitely generated.*

2.2 In this second subsection, we construct maximal chains of length 2^k of partial clones containing the set $\mathcal{I}$ on $\mathbf{k}$. We need the following terminology: if C_1 and C_2 are two partial clones, we say that C_2 *covers* C_1 if $C_1 \subset C_2$ and the inclusions $C_1 \subset D \subset C_2$ hold for no partial clone D on $\mathbf{k}$. Moreover we say that the chain of partial clones

$$C_1 \subset C_2 \subset \ldots \subset C_{n-1} \subset C_n$$

is a *maximal chain* if the partial clone C_i covers the partial clone C_{i-1} for all $i = 2, \ldots, n$. We start with the following general result.

Lemma 6 *Let C_1 and C_2 be two partial clones on $\mathbf{k}$ with $C_1 \subset C_2$. Suppose that there is a subset D of $\mathbf{k}$ and $a \in \mathbf{k} \setminus D$ such that*

1) $\bigcap_{x\in D\cup\{a\}} \mathrm{pPol}\,\{x\} \subseteq C_1 \subset C_2$ *and*

2) *for every $n \geq 1$ and every $f \in \mathrm{Par}^{(n)}(\mathbf{k})$, $f \in C_2^{(n)} \setminus C_1^{(n)}$ if and only if the following two conditions hold:*

i) $\forall\, x \in D \cup \{a\}$ $(x, \ldots, x) \in \mathrm{dom}\,(f)$, and
ii) $\forall\, d \in D$ $f(d, \ldots, d) = d$, and
iii) $f(a, \ldots, a) \neq a$.

Then the partial clone C_2 covers the partial clone C_1.

Proof. First notice that since $\bigcap_{x\in D\cup\{a\}} \mathrm{pPol}\,\{x\} \subseteq C_1 \subset C_2$, both C_1 and C_2 contain the set $\mathrm{Str}\,(\mathbf{J}(\mathbf{k}))$ of all partial projections on $\mathbf{k}$, and consequently they are both strong partial clones. Furthermore, if $D = \emptyset$, then $\bigcap_{x\in D\cup\{a\}} \mathrm{pPol}\,\{x\} = \mathrm{pPol}\,\{a\}$. As $\mathrm{pPol}\,\{a\}$ is a maximal partial clone on $\mathbf{k}$, we deduce in such a case that $C_1 = \mathrm{pPol}\,\{a\}$ and $C_2 = \mathrm{Par}(\mathbf{k})$. Let $m \geq 1$ and $f_0 \in C_2 \setminus C_1$ be m-ary. Then by assumption $\rho := \{(d, \ldots, d) \mid d \in D\} \cup \{(a, \ldots, a)\} \subseteq \mathrm{dom}\,(f_0)$. We show that $C_2 = \langle C_1 \cup \{f_0\}\rangle$. Since $C_1 \subseteq \langle C_1 \cup \{f_0\}\rangle$, we have that $\langle C_1 \cup \{f_0\}\rangle$ is a strong partial clone and so it contains the partial function $f_1 := f_0|_\rho$. Consequently the unary partial function f defined by $f := f_0[e_1^1, \ldots, e_1^1]$ belongs to $\langle C_1 \cup \{f_0\}\rangle$. Notice that $\mathrm{dom}\,(f) = D \cup \{a\}$ and as f_0 satisfies the conditions ii) and iii) above, $f(d) = d$ for all $d \in D$ and $f(a) \neq a$. Now let $g \in C_2 \setminus C_1$ be n-ary. We show that $g \in \langle C_1 \cup \{f_0\}\rangle$. As $k \geq 2$, we may assume w.l.o.g. that $a \neq 0$. Define three n-ary partial functions g_0, g_a and $\overline{g}$ by setting

$\mathrm{dom}\,(g_0) = \mathrm{dom}\,(g_a) = \mathrm{dom}\,(\overline{g}) = \mathrm{dom}\,(g)$ and, for $y \in \{0, a\}$

$$g_y(\underset{\sim}{x}) = \begin{cases} a, & \text{if } \underset{\sim}{x} = (a, \ldots, a) \\ d, & \text{if } \underset{\sim}{x} = (d, \ldots, d) \text{ and } d \in D \\ y & \text{otherwise,} \end{cases}$$

while

$$\overline{g}(\underset{\sim}{x}) = \begin{cases} a, & \text{if } \underset{\sim}{x} = (a, \ldots, a) \\ g(\underset{\sim}{x}) & \text{otherwise.} \end{cases}$$

Notice that from $g \in C_2 \setminus C_1$, we have $g(d, \ldots, d) = d$ and thus $\overline{g}(d, \ldots, d) = d$ for all $d \in D$. Now since $h(x, \ldots, x) = x$ for all $x \in D \cup \{a\}$ and all $h \in \{g_0, g_a, \overline{g}\}$, we have $\{g_0, g_a, \overline{g}\} \subset \bigcap_{x\in D\cup\{a\}} \mathrm{pPol}\,\{x\} \subseteq C_1$, hence $\{g_0, g_a, \overline{g}\} \subset C_1$.
Set $\overline{f} := f[g_a]$. Then $\overline{f} \in \langle C_1 \cup \{f_0\}\rangle$, $\mathrm{dom}\,(\overline{f}) = \mathrm{dom}\,(g)$ and

$$\overline{f}(\underset{\sim}{x}) = \begin{cases} d, & \text{if } \underset{\sim}{x} = (d, \ldots, d) \text{ and } d \in D \\ f(a) & \text{otherwise} \end{cases}$$

We need one more partial function. Let α be the 4-ary partial function defined by
$\mathrm{dom}\,(\alpha) := \{(a, a, f(a), a)\} \cup \{(d, d, d, d) \mid d \in D\} \cup \{(0, a, f(a), x) \mid x \in \mathbf{k}\}$,
and for $\underset{\sim}{x} := (x_1, x_2, x_3, x_4)$,

$$\alpha(\underset{\sim}{x}) = \begin{cases} g(a, \ldots, a), & \text{if } \underset{\sim}{x} = (a, a, f(a), a) \\ d, & \text{if } \underset{\sim}{x} = (d, \ldots, d) \text{ and } d \in D \\ x, & \text{if } \underset{\sim}{x} = (0, a, f(a), x) \text{ and } x \in \mathbf{k} \end{cases}$$

As $f(a) \neq a$, we have $(a, a, a, a) \notin \mathrm{dom}\,(\alpha)$ and moreover it is clear that $(x, x, x, x) \in \mathrm{dom}\,(\alpha) \iff x \in D$. Combining this with $\alpha(d, d, d, d) = d$ for all $d \in D$, we get that $\alpha \in \bigcap_{x\in D\cup\{a\}} \mathrm{pPol}\,\{x\} \subseteq C_1$, thus $\alpha \in C_1$. Now

we show that $g = \alpha[g_0, g_a, \overline{f}, \overline{g}]$. Set $\gamma := \alpha[g_0, g_a, \overline{f}, \overline{g}]$. In view of dom (g_0) = dom (g), we have dom $(\gamma) \subseteq$ dom (g). Conversely, using the definitions of $g_0, g_a, \overline{f}$ and $\overline{g}$, we can verify that

$$(g_0(\underset{\sim}{x}), g_a(\underset{\sim}{x}), \overline{f}(\underset{\sim}{x}), \overline{g}(\underset{\sim}{x})) \in \text{dom } (\alpha)$$

for all $\underset{\sim}{x} \in$ dom (g), thus dom (γ) = dom (g). Let $\underset{\sim}{x} \in$ dom (γ). Then

$$\gamma(\underset{\sim}{x}) = \begin{cases} \alpha(a,a,f(a),a) = g(a,\dots,a) \text{ if } \underset{\sim}{x} = (a,\dots,a) \\ \alpha(d,d,d,d) = d, \text{ if } \underset{\sim}{x} = (d,\dots,d) \text{ and } d \in D \\ \alpha(0,a,f(a),\overline{g}(\underset{\sim}{x})) = \overline{g}(\underset{\sim}{x}) \text{ otherwise} \end{cases}$$

and so by the definition of $\overline{g}$, $g(\underset{\sim}{x}) = \gamma(\underset{\sim}{x})$ for all $\underset{\sim}{x} \in$ dom (g), i.e., $g = \gamma \in \langle C_1 \cup \{f_0\}\rangle$. ∎

We now construct maximal chains of length 2^k of partial clones containing the set $\mathcal{I}$ of all partial idempotent functions on $\mathbf{k}$. We will use the following notation. If C and C' are two partial clones on $\mathbf{k}$ with $C \subseteq C'$, then the *interval of partial clones* $[C,\ C']$ is defined by

$$[C,\ C'] = \{D \mid D \text{ is a partial clone on } \mathbf{k} \text{ and } C \subseteq D \subseteq C'\}.$$

We start with an example to illustrate the general construction. Let $k = 4$. For notational ease set

$$C_4 := \bigcap_{X \in \{\{0\},\{1\},\{2\},\{3\}\}} \mathcal{P}_X \text{ and}$$

$$C_3 := \bigcap_{X \in \{\{0\},\{1\},\{2\}\}} \mathcal{P}_X,$$

therefore C_3 is a partial clone on $\{0,1,2,3\}$ that contain the set of all partial idempotent functions C_4 on $\{0,1,2,3\}$. We show that the interval of partial clones $[C_4, C_3]$ contains several unrefinable chains of partial clones of length 8 on $\{0,1,2,3\}$. Set

$\mathcal{F}_1 := \{\{0\},\{1\},\{2\},\{0,1,2,3\}\}$, $\mathcal{F}_2 := \mathcal{F}_1 \cup \{1,2,3\}$,
$\mathcal{F}_3 := \mathcal{F}_2 \cup \{0,2,3\}$, $\mathcal{F}_4 := \mathcal{F}_3 \cup \{0,1,3\}$,
$\mathcal{F}_5 := \mathcal{F}_4 \cup \{2,3\}$, $\mathcal{F}_6 := \mathcal{F}_5 \cup \{1,3\}$, $\mathcal{F}_7 := \mathcal{F}_6 \cup \{0,3\}$ and $\mathcal{F}_8 := \mathcal{F}_7 \cup \{3\}$.

The equality $C_4 = \bigcap_{X \in \mathcal{F}_8} \mathcal{P}_X$ is straightforward. Consider the chain of partial clones

$$C_4 \subset \bigcap_{X \in \mathcal{F}_7} \mathcal{P}_X \subset \bigcap_{X \in \mathcal{F}_6} \mathcal{P}_X \subset \dots \subset \bigcap_{X \in \mathcal{F}_2} \mathcal{P}_X \subset \bigcap_{X \in \mathcal{F}_1} \mathcal{P}_X \subset C_3. \quad (\star)$$

It is easy to verify that Lemma 6 applies to each pair of consecutive partial clones in the chain $(\star)$. Consider for example the two partial clones $\bigcap_{X \in \mathcal{F}_3} \mathcal{P}_X$ and $\bigcap_{X \in \mathcal{F}_2} \mathcal{P}_X$. Then clearly $\bigcap_{x \in \{0,2,3\}} \text{pPol } \{x\} \subseteq \bigcap_{X \in \mathcal{F}_3} \mathcal{P}_X$ and for any n-ary partial function f, $f \in \bigcap_{X \in \mathcal{F}_2} \mathcal{P}_X \setminus \bigcap_{X \in \mathcal{F}_3} \mathcal{P}_X$ iff $(0,\dots,0), (2,\dots,2), (3,\dots,3) \in$ dom (f) and $f(0,\dots,0) = 0$, $f(2,\dots,2) = 2$ and $f(3,\dots,3) \neq 3$. So take $D := \{0,2\}$, $a = 3$ and apply Lemma 6.

We now turn to the general case.

Let $2 \leq t \leq k-1$. Put $C_t := \bigcap_{X \in \{\{0\},\dots,\{t-1\}\}} \mathcal{P}_X$ and $C_{t+1} := \bigcap_{X \in \{\{0\},\dots,\{t-1\},\{t\}\}} \mathcal{P}_X$. We show that the interval of partial clones $[C_{t+1},\ C_t]$ contains maximal chains of size 2^t. Put

$$\mathcal{F}_{1,t+1} := \{\{0\},\dots,\{t-1\},\{0,1,\dots,t-1,t\}\}$$

and consider the partial clone $C_{1,t+1} := \bigcap_{X \in \mathcal{F}_{1,t+1}} \mathcal{P}_X$.

Then an n-ary partial function f satisfies

$$f \in C_t \setminus C_{1,t+1}$$

if and only if $f \in$ pPol $\{0\} \cap \dots \cap$ pPol $\{t-1\}$ and $f \notin$ pPol $\{(0,\dots,t-1,t)\}$. This holds if and only if $\{(0,\dots,0),\dots,(t,\dots,t)\} \subseteq$ dom (f), $f(d,\dots,d) = d$ for all $d \in \{0,\dots,t-1\}$ and $f(t,\dots,t) \neq t$. By Lemma 6 the partial clone C_t covers $C_{1,t+1}$ (here $D = \{0,\dots,t-1\}$ and $a = t$).

Starting with $\mathcal{F}_{1,t+1}$, we define recursively 2^t families of subsets of $\mathbf{t+1} := \{0,1,\dots,t-1,t\}$ as follows: in the first step we add to $\mathcal{F}_{1,t+1}$ one by one all subsets of the set $\mathbf{t+1}$ that are of size t and contain the element t. More precisely, let $Y_{1,t},\dots,Y_{t,t}$ be a list of all subsets of $\mathbf{t+1}$ of size t and containing the element t. Set $\mathcal{F}_{1,t} := \mathcal{F}_{1,t+1} \cup \{Y_{1,t}\},\dots,\mathcal{F}_{t,t} := \mathcal{F}_{t-1,t} \cup \{Y_{t,t}\}$. Next let $1 \leq j \leq t-1$ and suppose that each of $\mathcal{F}_{1,j+1},\dots,\mathcal{F}_{\binom{t}{j},j+1}$ is defined. We use the family of all subsets of $\mathbf{t+1}$ of size j that contain the element t. There are $\binom{t}{j-1}$ such sets, let $Y_{1,j},\dots,Y_{\binom{t}{j-1},j}$ be a list of them. Set

$$\mathcal{F}_{1,j} := \mathcal{F}_{\binom{t}{j},j+1} \cup \{Y_{1,j}\},\dots,$$

$$\mathcal{F}_{\binom{t}{j-1},j} := \mathcal{F}_{\binom{t}{j-1}-1,j} \cup \{Y_{\binom{t}{j-1},j}\}.$$

In this process, the second to the last family of subsets to be defined is $\mathcal{F}_{t,2}$, it actually consists of all subsets of $\mathbf{t+1}$ of size at least 2 and that contain the element t. Finally $\mathcal{F}_{1,1}$ is defined as the family of all subsets of $\mathbf{t+1}$ that contain the element t.

For every $j = 1,\dots,t$ and every $i = 1,\dots,\binom{t}{j-1}$, put $C_{i,j} := \bigcap_{X \in \mathcal{F}_{i,j}} \mathcal{P}_X$. Clearly the family of partial clones $\{C_{i,j}\}_{j=1,\dots,t,\ i=1,\dots,\binom{t}{j-1}}$ is contained in the interval $[C_{t+1},\ C_t]$. We show that it forms a maximal chains of partial clones. To see this let $j = 2,\dots,t-1$ and consider the two partial clones $C_{\binom{t}{j},j+1}$ and $C_{1,j}$. We use Lemma 6 to show that $C_{\binom{t}{j},j+1}$ covers $C_{1,j}$. W.l.o.g.

suppose we chose the set $Y_{1,j}$ to be $\{0,\ldots,j-2,t\}$. Clearly statement 1) of Lemma 6 holds for $C_{1,j}$ and $C_{\binom{t}{j},j+1}$, with $D=\{0,\ldots,j-2\}$ and $a=t$. Now let f be an n-ary partial function. Then $f \in C_{\binom{t}{j},j+1} \setminus C_{1,j}$ if and only if it preserves each j-ary relation on $\mathbf{k}$ of the form $\{(a_1,\ldots,a_{j-1},t)\}$ (where $a_1,\ldots,a_{j-1} \in \{0,\ldots,t-1\}$ are pairwise distinct), and, moreover f does not preserve the relation $\{(0,\ldots,j-2,t)\}$. This holds if and only if

i) $\forall\, x \in \{0,\ldots,j-2\} \cup \{t\}$ $(x,\ldots,x) \in \mathrm{dom}\,(f)$, and

ii) $\forall\, d \in \{0,\ldots,j-2\}$, $f(d,\ldots,d)=d$, and

iii) $f(t,\ldots,t) \neq t$.

By Lemma 6, $C_{\binom{t}{j},j+1}$ covers the partial clone $C_{1,j}$. We leave to the reader the proof that the partial clone $C_{i+1,j}$ covers the partial clone $C_{i,j}$ for every $j=1,\ldots,t$ and every $i=1,\ldots,\binom{t}{j}-1$. This proof is similar to the one given above, however here one has to specify the choices of the sets $Y_{\ell,j}$ for all $\ell=1,\ldots,i$.

Putting together the chains thus constructed in each of the intervals of partial clones $[C_{t+1},\ C_t]$ for all $2 \leq t \leq k-1$, we get a maximal chain of length $2^{k-1}+2^{k-2}+\ldots+2$. On the top of such a chain one adds the maximal chain $\mathrm{pPol}\,\{0\} \subset \mathrm{Par}(\mathbf{k})$ to obtain a maximal chain of length 2^k of partial clones containing the set of all idempotent partial functions $\mathcal{I}$ on $\mathbf{k}$.

Remark Very recently, D. Lau provided the first author with a proof of the following result: there are finitely many partial clones (strong or not) that contain the clone $\bigcap_{x\in\mathbf{k}} \mathrm{Pol}\,\{x\}$ of all idempotent functions on $\mathbf{k}$. As $\bigcap_{x\in\mathbf{k}} \mathrm{Pol}\,\{x\}$ is a finitely generated clone, it follows that every partial clone containing $\bigcap_{x\in\mathbf{k}} \mathrm{Pol}\,\{x\}$ is a finitely generated partial clone. Corollary 5 is a particular case of this result.

Acknowledgments We thank Ivo Rosenberg as well as the referee of this paper for their valuable suggestions.

References

[1] K. Baker, A. Pixley, Polynomial interpolation and the Chinese remainder theorem for algebraic systems, *Math. Z.* **143**, (1975) 165-174.

[2] F. Börner and L. Haddad, Maximal partial clones with no finite basis. To appear in *Algebra Universalis.*

[3] F. Börner and L. Haddad, Generating sets for clones and partial clones. *Proceedings* 28th IEEE *Internat. Sympos. Multiple-valued Logic*, Japan (1998), 363-368.

[4] J. Fugère and L. Haddad, Clones and partial clones containing all idempotent functions. *Proceedings* 28th IEEE *Internat. Sympos. Multiple-valued Logic*, Japan (1998), 369-373.

[5] D. Lau, Personnal communications.

[6] R. W. Quackenbush, On the composition of idempotent functions. *Algebra Universalis* **1**, (1971) 7-12.

[7] Romov, B. A., The algebras of partial functions and their invariants, *Kibernatika*; English translation in *Cybernetics* **17** (1981) 157-167.

[8] Rosenberg, I.G., *Composition of functions on finite sets, completeness and relations, a short survey,* in : D Rine, ed., Multiple-valued Logic and Computer Science (North-Holland, Amsterdam, 1977); 2nd edition (1984) 150-192.

Partial Clones and their Generating Sets

Lucien Haddad * Dietlinde Lau **

Abstract

We present some of our recent results on partial clones. Let A be a non singleton finite set. For every maximal clone C on A, we find the maximal partial clone on A that contains C. We also construct families of finitely generated maximal partial clones as well as a family of not finitely generated maximal partial clones on A. Furthermore, we study the pairwise intersections of all maximal partial clones of Słupecki type on A.

1 Introduction and Preliminaries

In the sequel $k \geq 2$ and A is a k-element set. For a positive integer n, an n-ary *partial function* on A is a map $f : \mathrm{dom}\,(f) \to A$ where $\mathrm{dom}\,(f)$ is a subset of A^n called the *domain* of f. Let $\mathcal{P}_A^{(n)}$ denote the set of all n-ary partial functions on A and let $\mathcal{P}_A := \bigcup_{n\geq 1} \mathcal{P}_A^{(n)}$. Moreover set $\mathcal{O}_A^{(n)} := \left\{ f \in \mathcal{P}_A^{(n)} \mid \mathrm{dom}\,(f) = A^n \right\}$ and $\mathcal{O}_A := \bigcup_{n\geq 1} \mathcal{O}_A^{(n)}$. $\mathcal{O}_A$ is the set of all *functions* on the set A.

For $n, m \geq 1$, $f \in \mathcal{P}_A^{(n)}$ and $g_1, \ldots, g_n \in \mathcal{P}_A^{(m)}$, the *composition* of f and $g_1, \ldots, g_n$, denoted $f[g_1, \ldots, g_n]$ is the m-ary partial function on A defined by

$$\mathrm{dom}\,(f[g_1, \ldots, g_n]) := \{\vec{a} \in A^m \mid \vec{a} \in \bigcap_{i=1}^{n} \mathrm{dom}\,(g_i) \text{ and } (g_1(\vec{a}), \ldots, g_n(\vec{a})) \in \mathrm{dom}\,(f)\};$$

and moreover

$$f[g_1, \ldots, g_n](\vec{a}) := f(g_1(\vec{a}), \ldots, g_n(\vec{a}))$$

for all $\vec{a} \in \mathrm{dom}\,(f[g_1, \ldots, g_n])$.

For every positive integer n and each $1 \leq i \leq n$, let e_i^n denote the *n-ary i-th projection* defined by $\mathrm{dom}\,(e_i^n) = A^n$ and $e_i^n(a_1, \ldots, a_n) = a_i$ for all $(a_1, \ldots, a_n) \in A^n$. Furthermore let

$$J_A := \{e_i^n \mid 1 \leq i \leq n < \infty\}$$

be the set of all projections.

Definitions A *partial clone* on A is a composition closed subset of $\mathcal{P}_A$ containing J_A. If a partial clone C is contained in the set of all functions $\mathcal{O}_A$, then it is called *a clone* on A.

A partial clone (a clone) C is said to be a *maximal partial clone* (*maximal clone*) if $C \subset C_0 \subset \mathcal{P}_A$ ($C \subset C_0 \subset \mathcal{O}_A$) for no partial clone C_0 (for no clone C_0) on A.

I. G. Rosenberg described all maximal clones on A in [21], (see also [22] and [23]). The description is in terms of polymorphisms of relations, these are defined as follows.

Let $h \geq 1$, ϱ be an h-ary relation on A (i.e., a subset of A^h) and f be an n-ary partial function on A. Denote by $\mathcal{M}(\mathrm{dom}\,(f), \varrho)$ the set of all $h \times n$ matrices M on A whose columns $M_{*j} \in \varrho$, for $j = 1, \ldots, n$ and whose rows $M_{i*} \in \mathrm{dom}\,(f)$ for $i = 1, \ldots, h$. We say that the partial function f *preserves the relation* ϱ if for every $M \in \mathcal{M}(\mathrm{dom}\,(f), \varrho)$, $(f(M_{1*}), \ldots, f(M_{h*})) \in \varrho$.

Set $\mathrm{pPol}\,\varrho := \{f \in \mathcal{P}_A \mid f \text{ preserves } \varrho\}$ and $\mathrm{Pol}\,\varrho := \mathrm{pPol}\,\varrho \cap \mathcal{O}_A$ (i.e., $\mathrm{Pol}\,\varrho$ is the set of all functions that preserve the relation ϱ). It is well known and easy to show (see e.g., [12], [17] and [21]) that $\mathrm{pPol}\,\varrho$ ($\mathrm{Pol}\,\varrho$) is a strong partial clone on A (a clone on A) called the partial clone *determined by* ϱ (the clone *determined by* ϱ).

A partial clone C is said to be *strong* if it contains all subfunctions of its functions, i.e., if for every $f \in C$ and $g \in \mathcal{P}_A$, $[g = f|_{\mathrm{dom}\,(g)} \Longrightarrow g \in C]$.

For every set $F \subset \mathcal{P}_A$, let $\mathrm{Str}\,(F)$ denote the *strong closure* of F, i.e.,

$\mathrm{Str}\,(F) := \{g \in \mathcal{P}_A \mid g \text{ is a subfunction of some } f \in F\}$

Furthermore let $\langle F \rangle$ denote the *partial clone generated by* F, i.e., the intersection of all partial clones on A containing the set F. If $C = \langle F \rangle$, then we say that F is a *generating set* for C. A partial clone C is said to be *finitely generated* if it admits a finite generating set; i.e., if $C = \langle F \rangle$ for some finite set $F \subseteq C$. Finally, a

*Département de mathématiques et informatique, collège militaire royal du Canada, B. P. 17000, STN Forces, Kingston, ON, K7K 7B4 Canada. Email haddad-l@rmc.ca

**Fachbereich Mathematik, Universität Rostock, Universitätsplatz 1, 18055 Rostock, Germany. Email dietlinde.lau@mathematik.uni-rostock.de

0-7695-0161-3/99 $10.00 © 1999 IEEE

partial clone C is called of *Słupecki type* if C contains the set $\mathcal{O}_A^{(1)}$ of all unary functions on A.

The purpose of this paper is to present the recent results obtained by the two authors in [6] and [7]. As many of the results in [6] are based on Rosenberg's classification of all maximal clones on A, we need to state Rosenberg's Theorem. Let $h \geq 2$ be an integer. Put

$\tau_h := \{(a_1, \ldots, a_h) \in A^h \mid a_i = a_j$ for some $1 \leq i < j \leq h\}$

If ϱ is an h-ary relation on A and S_h denotes the set of all permutations on $\{1, \ldots, h\}$, then for $\pi \in S_h$ let

$$\varrho^{(\pi)} := \{(x_{\pi(1)}, \ldots, x_{\pi(h)}) \mid (x_1, \ldots, x_h) \in \varrho\}.$$

An h-ary relation ϱ is said to be

1. *totally symmetric* (*symmetric* in the case $h = 2$) if $\varrho^{(\pi)} = \varrho$ for every $\pi \in S_h$ i.e., if
$$(x_1, \ldots, x_h) \in \varrho \Longleftrightarrow (x_{\pi(1)}, \ldots, x_{\pi(h)}) \in \varrho$$
for all $\pi \in S_h$.
2. *totally reflexive* (*reflexive* in the case $h = 2$) if $\tau_h \subseteq \varrho$.
3. *prime affine*, if $h = 4$ and there are a prime number p and an integer $m \geq 1$ such that $k = p^m$ and
$$\varrho := \{(a_1, a_2, a_3, a_4) \in A^4 \mid a_1 + a_2 = a_3 + a_4\},$$
where $(A, +)$ is an elementary Abelian p-group (i.e., $\underbrace{x + x + \ldots + x}_{p \text{ times}} = 0$ for every element $x \in A$, where 0 is the identity element of the group).
4. *central*, if $\varrho \neq A^h$, ϱ is totally symmetric, totally reflexive and, if $h \geq 2$, there exists an element $c \in A$ such that $\{c\} \times A^{h-1} \subseteq \varrho$. (The element c is called a *central element* of ϱ.)
5. *elementary*, if $k = h^m$, $m \geq 1$, $h \geq 3$ and
$(a_1, a_2 \ldots, a_h) \in \varrho \Longleftrightarrow$
$$\forall i \in \{0, \ldots, m-1\}\ (a_1^{(i)}, a_2^{(i)}, \ldots, a_h^{(i)}) \in \tau_h,$$
where $a^{(i)}$ ($a \in \{0, 1, \ldots, h^m - 1\}$) denotes the i-th digit in the expansion
$a = a^{(m-1)} \cdot h^{m-1} + a^{(m-2)} \cdot h^{m-2} + \ldots + a^{(1)} \cdot h + a^{(0)}$.
6. a *homomorphic inverse image of an h-ary relation ϱ' on A'*, if there exists a surjective mapping $q : A \longrightarrow A'$ with
$$(a_1, \ldots, a_h) \in \varrho \Longleftrightarrow (q(a_1), \ldots, q(a_h)) \in \varrho'$$
for all $a_1, \ldots, a_h \in A$.
7. *h-universal*, if ϱ is a homomorphic inverse image of an h-ary elementary relation.

Furthermore, let

$\mathcal{C}_A$ be the set of all h-ary central relations on A, $1 \leq h \leq k-1$;

$\mathcal{U}_A$ be the set of all non trivial equivalence relations on A ;

$\mathcal{S}_A := \bigcup_{p,\ p \mid k} \mathcal{S}_{k,p}$, where $\mathcal{S}_{k,p} := \{\{(x, s(x)) \mid x \in A\} \mid s \in P_{k,p}\}$ and $P_{k,p}$ is the set of all permutations on A consisting of cycles of the same prime length p ;

$\mathcal{M}_A$ be the set of all bounded partial order relations on A;

$\mathcal{L}_A$ be the set of all prime affine relations on A,

$\mathcal{B}_A$ be the set of all h-universal relations, $3 \leq h \leq k$.

Here is Rosenberg's Theorem.

Theorem 1 [23] *Let $k \geq 2$ and A be a k-element set. Every proper clone on A extends to a maximal one. If M is a clone on A, then M is a maximal clone if and only if $M = \mathrm{Pol}\ \varrho$, for some $\varrho \in \mathcal{C}_A \cup \mathcal{M}_A \cup \mathcal{S}_A \cup \mathcal{U}_A \cup \mathcal{L}_A \cup \mathcal{B}_A$.* □

In the sequel $\mathbf{M}$ stands for the set of all relations that determine maximal clones on A. Thus by Theorem 1, $\mathbf{M} := \mathcal{C}_A \cup \mathcal{M}_A \cup \mathcal{U}_A \cup \mathcal{S}_A \cup \mathcal{L}_A \cup \mathcal{B}_A$.

More recently, the maximal partial clones on A were described by several authors. We refer the reader to [6], [11], [12], [13] or [18] for a description of all maximal partial clones on A.

Two problems are addressed in [6], they both concern the partial clones pPol ϱ where $\varrho \in \mathbf{M}$ is one of the relations in Rosenberg's Theorem. First, is pPol ϱ a maximal partial clone on A and if not which maximal partial clone on A contain Pol ϱ? Second, is pPol ϱ a finitely generated partial clone on A?

In [7], intersections of maximal partial clones of Słupecki type on A are studied. It is shown that with one exception, the intersection of two strong maximal partial clones of Słupecki type on A is covered by both those two maximal partial clones. Furthermore, it is shown that the situation is quite different if the unique non-strong maximal partial clone of Słupecki type on A is involved in the intersection.

2 Partial clones containing maximal clones

Unless otherwise specified, all results presented in this section are established in [6]. Let M be a maximal clone on A. Since every partial clone on A extends to a maximal one (see [10]), it is natural to ask how many maximal partial clones on A contain the clone M? The following is shown using a result established by Freivald in [5]:

Proposition 2 *Let M be a maximal clone on A. Then M is contained in exactly one maximal partial clone on A.* □

It follows that for every $\varrho \in \mathbf{M}$, either pPol ϱ is a maximal partial clone on A or there is a unique maximal partial clone on A that contains pPol ϱ. The result below shows that of the six families of relations that determine maximal clones, four determine maximal partial clones on A.

Proposition 3 *Let $M_2 := \mathcal{S}_A \setminus \mathcal{S}_{k,2}$ and $\varrho \in \mathbf{M} \setminus (\mathcal{L}_A \cup M_2)$. Then* pPol ϱ *is a maximal partial clone on A. Furthermore, if $k = 2^m$ for some $m \geq 1$, then* pPol ϱ *is a maximal partial clone on A for every $\varrho \in \mathcal{L}_{2^m}$.* □

The relations in $\mathcal{S}_{k,p} \cup \mathcal{L}_A$ determine maximal clones but do not determine maximal partial clone on A. Now given $\varrho \in \mathcal{S}_{k,p} \cup \mathcal{L}_A$, one can find the unique maximal partial clone on A that contain pPol ϱ. Starting with $\varrho_s := \{(x, s(x)) \mid x \in A\} \in \mathcal{S}_{k,p}$, we have

Proposition 4 *Let $p \geq 3$ be a prime number, s be a permutation on A consisting of cycles of the same prime length p and $\varrho_{s,p} := \{(x, s(x), s^2(x), \ldots, s^{p-1}(x)) \mid x \in A\}$. Then* pPol $\varrho_s \subset$ pPol $\varrho_{s,p}$ *and* pPol $\varrho_{k,p}$ *is a maximal partial clone on A.* □

We turn to the family $\mathcal{L}_A$ of all prime affine relations on A. Let λ be such a relation. The maximal clone Pol λ is described in [22] (see also [6]). As mentioned above, the quaternary relation λ is defined in terms of a binary function $+$ on A such that $(A, +)$ is an elementary Abelian p-group. It is well known that this implies that
$(A, +) \cong (\mathbf{p}, +) \times \ldots \times (\mathbf{p}, +)$, where $\mathbf{p} := \{0, \ldots, p-1\}$ and $(\mathbf{p}, +)$ is the cyclic group mod p. W.l.o.g assume $A = \mathbf{p}^m$ and consider the p-ary relation λ_p defined on $\mathbf{p}^m$ by
$\lambda_p := \{(\vec{a}, \vec{a} + \vec{b}, \vec{a} + 2 \cdot \vec{b}, \ldots, \vec{a} + (p-1) \cdot \vec{b}) \mid \vec{a}, \vec{b} \in \mathbf{p}^m\}$,
then

Proposition 5 pPol λ_p *is a maximal partial clone on $A = \mathbf{p}^m$ that properly contains the partial clone* pPol λ. □

The rest of this section is devoted to discuss generating sets for the partial clones determined by the relations in the set $\mathbf{M}$. We refer the reader to [2] for a brief survey on generating sets for clones.
The second author has shown in [14] that the maximal clone Pol ϱ is finitely generated on A for all $\varrho \in \mathbf{M} \setminus \mathcal{M}_A$. In view of this result, it is natural to ask whether the partial clone pPol ϱ for $\varrho \in \mathbf{M}$ is finitely generated. The methods used to tackle this question come from [2] (see also [3]). A slight modification of the concept of a separating clone, introduced in [2], is given in [6].

Definition Let $r \geq 1$ be an integer. A clone C on A is an *r-separating* clone if for all $n \geq 1$, there exist $g_1, \ldots, g_r \in C \cap \mathcal{O}_A^{(n)}$ such that the map $g : A^n \to A^r$ defined by $\vec{a} \to (g_1(\vec{a}), \ldots, g_r(\vec{a}))$ satisfies $g^{-1}(g(\vec{b})) = \{\vec{b}\}$ for all $\vec{b} \in A^n$, (i.e., g takes the value $g(\vec{b})$ only once). Moreover, a clone is a *separating* clone if it is an r-separating clone for some $r \geq 1$ ([2], [3]).

Of the six families of maximal clones on A, five consist of separating clones. Let $\mathcal{M}_A^\star$ be the set of all binary relations ϱ on A such that (A, ϱ) is a lattice. We have

Lemma 6 *For all $\varrho \in \mathcal{C}_A \cup \mathcal{U}_A \cup \mathcal{S}_A \cup \mathcal{M}_A \cup \mathcal{M}_A^\star \cup \mathcal{B}_A$, there is an $r \in \{1, 2\}$ such that* Pol ϱ *is an r-separating clone on A.* □

Remark The Lemma above leaves out the class of relations $\mathcal{L}_A$. In fact it is shown in [6], Lemma 17, that the maximal clone Pol ϱ is not a separating clone for all $\varrho \in \mathcal{L}_A$.

The concept of r-separating clone is quite useful to determine whether a partial clone of the form Str C is finitely generated, where C is a clone on A. We need the following terminology. If a clone C is finitely generated, then the *order* of C, written ord (C), is the minimal positive integer r such that $\langle C^{(r)} \rangle = C$. Using results from [2], one can show

Lemma 7 *Let $r \geq 1$ and C be a finitely generated r-separating clone on A. Then the partial clone* Str (C) *is finitely generated and* ord (Str (C)) $\leq max\{$ord $(C), r\}$. □

Now in order to apply the two Lemmas above on the clones determined by the relations in Rosenberg's Theorem, one must examine the relations $\varrho \in \mathbf{M}$ for which the equality Str (Pol ϱ) = pPol ϱ holds. Almost all of the following result is established by Romov in [20]. However the proofs given in [20] rely on relational constructions as well as on many different results. Set $\mathcal{S}_A^\star := \mathcal{S}_{k,2} \cup \mathcal{S}_{k,3}$. We have:

Lemma 8 (a) $\forall \varrho \in \mathcal{C}_A \cup \mathcal{U}_A \cup \mathcal{S}_A^\star \cup \mathcal{M}_A^\star$, Str (Pol ϱ) = pPol ϱ *and*
(b) $\forall \varrho \in (\mathcal{S}_A \setminus \mathcal{S}_A^\star) \cup (\mathcal{M}_A \setminus \mathcal{M}_A^\star) \cup \mathcal{L}_A$, Str (Pol ϱ) $\neq$ pPol ϱ. □

The proof of Lemma 8 given in [6] is direct in the following sense: it is shown that for every $\varrho \in \mathcal{C}_A \cup \mathcal{U}_A \cup \mathcal{S}_A^\star \cup \mathcal{M}_A^\star$ and every partial function $f \in \mathrm{pPol}\,\varrho$, f can be extended to a function in Pol ϱ. Also, it is shown that for every $\varrho \in (\mathcal{S}_A \setminus \mathcal{S}_A^\star) \cup (\mathcal{M}_A \setminus \mathcal{M}_A^\star) \cup \mathcal{L}_A$, there is a function $g \in \mathrm{pPol}\,\varrho$ that cannot be extended to a function in Pol ϱ.

We mentioned earlier that the second author has shown that five of the six families of relations in Rosenberg's Theorem determine finitely generated maximal clones on A. The class of all bounded order is left out, in fact Tardos has provided in [26] a bounded order relation $\preceq$ on an 8-element set for which the maximal clone Pol $\preceq$ is not finitely generated. Nozaki and Lashkia have shown recently that a partial clone determined by any order relation is finitely generated, indeed

Theorem 9 ([15]) *Let $\varrho \in \mathcal{M}_A$. Then the maximal partial clone* pPol ϱ *has order* 2. □

Theorem 9 provides a family of finitely generated partial clones on A. There are more such families as the following result shows. For $2 \le h \le k$, let $\mathcal{C}_A^h$ denote the set of all h-ary central relations on A. We have

Theorem 10 a) *Let $|A| = 2$. If $\varrho \in \mathcal{S}_A$ then* ord $(\mathrm{pPol}_A \varrho) = 3$ *and if $\varrho \in \mathcal{C}_A$, then* ord $(\mathrm{pPol}_A \varrho) = 2$.

b) *Let $|A| = k \ge 3$.*

i) *If $\varrho \in \mathcal{U}_A \cup \mathcal{S}_A^\star \cup \mathcal{C}_A^1 \cup \mathcal{C}_A^2 \cup \mathcal{M}_A^\star$, then* ord (pPol ϱ) = 2, *and*

ii) *if $3 \le h \le k-1$ and $\varrho \in \mathcal{C}_A^h$, then* ord (pPol ϱ) $\le h$, *and*

iii) *if $\varrho \in \mathcal{M}_A \cup \mathcal{B}_A$, then* ord (Str (Pol ϱ)) = 2. □

Some of the partial clones in Theorem 10 are maximal. In fact a combination of Proposition 3, Theorem 9 and Theorem 10 gives

Theorem 11 *Let $k \ge 3$, A be a k-element set and $\varrho \in \mathcal{U}_A \cup \mathcal{C}_k \cup \mathcal{M}_A$. Then* pPol ϱ *is a finitely generated maximal partial clone on A.* □

One more family of finitely generated maximal partial clones on A is presented in [6], indeed

Theorem 12 *Let $p \ge 3$ be a prime number, $m \ge 1$, A be a pm-element set, s be a permutation with cycles of the same prime length p on A and $\varrho_s := \{(a, s(a), \ldots, s^{p-1}(a)) \mid a \in A\}$. Then the maximal partial clone* pPol ϱ_s *has order* 2. □

The problem of deciding whether pPol ϱ is a finitely generated partial clone for ϱ one of the relations in Rosenberg's Theorem is also settled in [6] for the family $\mathcal{L}_A$ of all prime affine relations on A. Here, we make use of the following criterion established in [2]:

Theorem 13 ([2], see also [3]) *Let A be a non-singleton finite set and C be a strong partial clone on A. If $C \cap \mathcal{O}_A$ is not a separating clone, then C is not finitely generated on A.* □

Now it is shown in [6] that the maximal clone Pol λ is not a separating clone for any $\lambda \in \mathcal{L}_A$, consequently

Theorem 14 *Let C be a strong partial clone with $C \cap \mathcal{O}_A =$ Pol λ for some $\lambda \in \mathcal{L}_A$. Then C is not finitely generated.* □

We point out here that a direct proof to Theorem 14, without using the criterion in Theorem 13, is presented in [6].

We conclude this section with the following remark. A result similar to Theorem 14 for the family of relations $\mathcal{B}_A$ is missing. Let $\varrho \in \mathcal{B}_A$. It is shown in [20] that Str (Pol ϱ) $\neq$ pPol ϱ, and Lemma 6 shows that Pol ϱ is a separating clone. Thus neither Theorem 10 nor Theorem 13 applies for pPol ϱ. We have a feeling that pPol ϱ is a not finitely generated maximal partial clone on A.

3 Intersections of Słupecki type maximal partial clones

In this section we present the results of [7]. If C and C' are two partial clones on A with $C \subseteq C'$, then by the *interval of partial clones* $[C, C']$ we mean the set $\{D \mid D$ is a partial clone on A and $C \subseteq D \subseteq C'\}$. As defined earlier, a partial clone is of *Słupecki type* if it contains the set $\mathcal{O}_A^{(1)}$ of all unary functions on A. For example, if $p_n \in \mathcal{P}_A^{(n)}$ denotes the n-ary partial function with empty domain, then

$$\Omega_A := \mathcal{O}_A \cup \{p_n \mid n \ge 1\},$$

is a Słupecki type partial clone on A. The definition of a Słupecki type partial clone is motivated by the well known Słupecki criterion, which states that Pol τ_k is the unique maximal clone that contains the set $\mathcal{O}_A^{(1)}$ of all unary functions on A (the relations τ_h have been defined in Section 1). Burle improved this result in [4] by proving that the interval of all clones of Słupecki type on A is simply the chain

$$\langle \mathcal{O}_A^{(1)} \rangle \subset \mathrm{Pol}\, R_1 \subset \mathrm{Pol}\, \tau_3 \subset \ldots \subset \mathrm{Pol}\, \tau_k \subset \mathcal{O}_A,$$

where

$$R_1 = \{(x,y,z,t) \in A^4 \mid [x = y \text{ and } z = t] \text{ or } [x = z \text{ and } y = t] \text{ or } [x = t \text{ and } y = z]\}.$$

As $\mathcal{O}_A^{(1)}$ is a finite set, one can easily obtain (e.g., see [2] and [3]) that every clone of Słupecki type on A is finitely generated.

The situation is quite different in the partial case. It is shown in [12] (and independently in [17]) that

there are $k+1$ Słupecki type maximal partial clones on the k-element set A, namely pPol R_1, pPol R_2, pPol $\tau_3, \ldots,$ pPol τ_k and Ω_A, where
$R_2 = \{(x,y,z,t) \in A^4 \mid [x = y \text{ and } z = t] \text{ or } [x = t \text{ and } y = z]\}$.
Notice that of all these $k+1$ maximal partial clones, only Ω_A is a non strong partial clone. Furthermore, it is shown in [8] (see also [3]) that the family of all Słupecki type partial clones on A is of continuum cardinality. Finally, none of the maximal partial clones pPol R_1, pPol R_2, pPol $\tau_3, \ldots,$ pPol τ_k is finitely generated is shown in [2] (see also [3]).
The conditions to recognize finitely generated partial clones established in [2] apply for partial clones C whose "$\mathcal{O}_A$ components" $C \cap \mathcal{O}_A$ are either non separating or not finitely generated clones. Let $C_1 \neq C_2$ be two Słupecki type maximal partial clones on A. Then it is easy to see that the clone $(C_1 \cap C_2) \cap \mathcal{O}_A$ is a separating finitely generated clone, and so the results in [2] cannot be used to decide whether the partial clone $C_1 \cap C_2$ is finitely generated. This is solved in [7]. We say that a partial clone C' *covers* a partial clone C on A if $C \subset C'$ and $[C, C'] = \{C, C'\}$. One of the main results in [7] is

Theorem 15 *Let $k \geq 2$, A be a k-element set and $C \neq C'$ be two strong maximal partial clones of Słupecki type on A. If $(C, C') \neq$ (pPol R_1, pPol R_2), then the partial clone $C \cap C'$ is covered by the maximal partial clone C'.* □

Theorem 15 is shown by several steps. A main tool used in the proof is the Definability Lemma established in [17] (see [9] and, for a weak version of it, see [12]).
The result of Theorem 15 above does not hold for $C =$ pPol R_1 and $C' =$ pPol R_2. Indeed it is shown in [7] that there is at least one partial clone that strictly lies in the interval of partial clones [pPol $R_1 \cap$ pPol R_2, pPol R_2]. Now even though the size of the interval of partial clones [pPol $R_1 \cap$ pPol R_2, pPol R_2] is not known, a combination of Theorem 15 with the fact that none of the k strong maximal partial clones of Słupecki type on A is finitely generated gives the following

Theorem 16 *Let $k \geq 2$, A be a k-element set and $C \neq C'$ be two strong maximal partial clones of Słupecki type on A. Then the partial clone $C \cap C'$ is not finitely generated on A.* □

The main purpose of [7] is to study the pairwise intersections of Słupecki type maximal partial clones on A. Since Ω_A is one of them, its intersection with all other Słupecki type partial clones is studied in that paper. In fact this question reduces to the following: let $\varrho \in \{R_1, R_2, \tau_3, \ldots, \tau_k\}$. Describe the interval of partial clones [Pol ϱ, pPol ϱ] on A. This type of problem has been tackled by the authors of [1], [24] and [25] for $A = \{0,1\}$, as well as by the first author in [8].
One of the results shown in [1] is that the interval of partial clones [Str (Pol R_1), pPol R_1] is of continuum cardinality on $\{0,1\}$. For $\ell \geq 2$, let
[Str (Pol R_1), pPol $R_1]_\ell :=$
$\{C \mid C$ is a partial clone on the ℓ-element set A and Str (Pol R_1) $\subseteq C \subseteq$ pPol $R_1\}$.
A one-to-one map
ϕ : [Str (Pol R_1), pPol $R_1]_2 \to$ [Str (Pol R_1), pPol $R_1]_k$ is constructed in [7] for every $k \geq 3$. As [Str (Pol R_1), pPol R_1] $\subseteq$ [$\Omega_A \cap$ Pol R_1, pPol R_1], we obtain

Proposition 17 *Let $k \geq 2$ and A be a k-element set. Then the interval of partial clones [$\Omega_A \cap$ pPol R_1, pPol R_1] has the cardinality of continuum on A.* □

Using families of relations defined in [13], the first author shows in [8] (Theorem 4) a result similar to Proposition 17 for the relation R_2.
We turn to the family of relations $\{\tau_3, \ldots, \tau_k\}$. For $n \geq 2$ and $h \geq 3$, let σ_{2n+h} be the relation of arity $(2n+h)$ on the set $\{1, \ldots, 2n+h\}$ defined by
$(x_1, \ldots, x_{2n+h}) \in \sigma_{2n+h} \iff |\{x_1, \ldots, x_{2n+h}\}| \leq h-1$ or $|\{x_1, \ldots, x_{2n+h}\}| = h$ with 1) $h-2$ symbols occurring each once and 2) one symbol occurring exactly twice and 3) one symbol occurring $2n$ times in $x_1, \ldots, x_{2n+h}$.
The inclusions $\Omega_A \cap$ pPol $\tau_h \subseteq$ pPol $\sigma_{2n+h} \subseteq$ pPol τ_h, for every $n \geq 2$ and every $3 \leq h \leq k$ are shown in [7]. The same paper gives a family $\{\phi_{2m+h} : m \geq 2\}$ of partial functions on A such that for n, $m \geq 2$, $\phi_{2m+h} \in$ pPol $\sigma_{2n+h} \iff n \neq m$. Thus

Proposition 18 *Let $k \geq 3$, A be a k-element set and $3 \leq h \leq k$. Then the interval of partial clones [$\Omega_A \cap$ pPol τ_h, pPol τ_h] has the cardinality of continuum on A.* □

A combination of the results above gives

Theorem 19 *Let $k \geq 2$, A be a k-element set and* pPol ϱ *be any maximal partial clone of Słupecki type on A. Then the interval of partial clones [$\Omega_A \cap$ pPol ϱ, pPol ϱ] has the cardinality of the continuum on A.* □

Remark A direct consequence of Theorem 19 is that for $|A| \geq 3$ and every
$\varrho \in \{R_1, R_2, \tau_3, \ldots, \tau_k\}$, the interval of partial clones [Pol ϱ, pPol ϱ] is of continuum cardinality on A. So one may ask the following general problems
1) let $\varrho \in \mathbf{M}$ be one of the relations that determine a maximal clone on A. Determine the interval of partial clones [Pol ϱ, pPol ϱ] on A. Dually
2) let ϱ be a relation such that pPol ϱ is a maximal partial clone on A. Determine the interval of partial clones [Pol ϱ, pPol ϱ] on A.

References

[1] V. B. Alekzeev,and L. L. Voronenko, Some closed classes in the partial two-valued logic (Russian). *Dis. Math.*, **6**, 4 (1994), 58 – 79

[2] F. Börner and L. Haddad, Maximal Partial Clones with no finite basis, preprint 1997, to appear in *Algebra Universalis.*

[3] F. Börner and L. Haddad, Generating sets for Clones and Partial Clones. *Proceedings* 28th IEEE *Internat. Sympos. Multiple-valued Logic*, Japan (1998), 363–368.

[4] G. A. Burle, Classes of k-valued logic which contain all functions of a single variable, (Russian), *Diskret. Analiz* (Novosibirsk) **10** (1967), 3–7.

[5] R.V. Freivald, Functional completeness for not everywhere defined functions of the algebra of logic, (in Russian). *Diskretn. Anal.*, Novosibirsk, **8** (1966) 55–68.

[6] L. Haddad and D. Lau, Families of finitely generated maximal partial clones. Preprint 1998, 30 pages.

[7] L. Haddad and D. Lau, Pairwise intersections of Słupecki type maximal partial clones. Preprint 1998, 28 pages.

[8] L. Haddad, On the depth of the intersection of two maximal partial clones, *MVL, an International Journal*, (1997) to appear.

[9] J. Fugère and L. Haddad, Clones and partial clones containing all idempotent functions. *Proceedings* 28th IEEE *Internat. Sympos. Multiple-valued Logic*, Japan (1998), 369-373.

[10] L. Haddad, I. G. Rosenberg and D. Schweigert, A maximal partial clone and a Słupecki-type criterion, *Acta Sci. Math.*, **54** (1990), 89–98.

[11] L. Haddad, and I.G. Rosenberg, Partial Sheffer functions *Europ J. Combinatorics* **12** (1991) 375-379.

[12] L. Haddad, and I.G. Rosenberg, Completeness theory for finite partial algebras *Algebra Universalis* **29** (1992) 378-401.

[13] L. Haddad and I. G. Rosenberg, Partial clones containing all permutations, *Bull. Austral. Math. Soc.*, **52** (1995), 263-278.

[14] D. Lau, Bestimmung der Ordnung maximaler Klassen von Funktionen der k-wertigen Logik. *Z. Math. Logik u. Grundl. Math.* **24** (1978), 79–96.

[15] A. Nozaki, and V. Lashkia, A finite basis of the set of all monotone partial functions defined over a finite poset. *Proceedings* 28th IEEE *Internat. Sympos. Multiple-valued Logic*, Japan (1998), 372–375.

[16] R. Pöschel and L. A. Kalužnin. *Funktionen-und Relationenalgebren.* VEB Deutscher Verlag der Wissenschaften, Berlin 1979.

[17] B. A. Romov, Maximal subalgebras of algebras of partial multivalued logic functions, *Kibernetika*; English translation in *Cybernetics* **1** (1980) 31-41.

[18] B. A. Romov, The completeness problem in the algebra of partial functions of finite-valued logic, *Kibernetika*, English translation in *Cybernetics* **26** (1990) 133-138.

[19] B. A. Romov, Maximal finitely defined subalgebras of partial functions of infinite-valued logic, *Kibernetika*, English translation in *Cybernetics*, (1993) 1-11.

[20] B. A. Romov, Extension of not everywhere defined functions of many-valued logic. *Kibernetika*, English translation in *Cybernetics* **3** (1987), 319–327.

[21] I.G. Rosenberg, La structure des fonctions de plusieurs variables sur un ensemble fini, *C. R. Acad. Sci. Paris* Sér A-B, **260** (1965) 3817-3819.

[22] I. G. Rosenberg, Über die funktionale Vollständigkeit in den mehrwertigen Logiken. *Rozpravy Československe Akad. Ved. Řada Mat. Přirod.* Ved **80** (1979), 3–93.

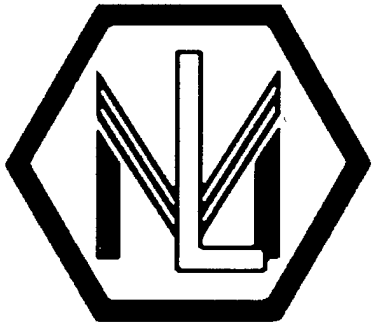
[23] I. G. Rosenberg, Composition of functions on finite sets, completeness and relations, a short survey. In D. Rine (ed.) *Multiple-valued Logic and Computer Science*, 2nd edition, North-Holland, Amsterdam (1984), 150-192.

[24] B. Strauch, On partial classes containing all monotone and zero-preserving total Boolean functions. *Math. Log. Quart.* **43** (1997).

[25] B. Strauch, The classes which contain all monotone and idempotent total Boolean functions, Universität Rostock, preprint 1996.

[26] G. Tardos, A maximal clone of monotone functions which is not finitely generated. *Order* **3** (1986), 211–218.

SESSION IVA
LOGIC DESIGN
CHAIR: Claudio Moraga



Evaluation of m-valued Fixed Polarity Generalizations of Reed-Muller Canonical Form

Elena Dubrova
Electronic System Design Lab
Department of Electronics
Royal Institute of Technology
S-164 40 Kista, Sweden
elena@ele.kth.se

Abstract

This paper compares the complexity of three different fixed polarity generalizations of Reed-Muller canonical form to multiple-valued logic: the Galois Field-based expansion introduced by Green and Taylor, the Reed-Muller-Fourier form of Stanković and Moraga, and the expansion over addition modulo m, minimum and the set of all literal operators introduced by the author and Muzio. An algorithm for computing the minimal canonical forms for these generalizations is implemented and applied to a set of encoded 4-valued benchmark functions, 3- and 4-valued adders and multipliers. The experimental results show that, for the benchmark functions, the Reed-Muller-Fourier form and our expansion yield a comparable number of products on average. They have 40% less products on average than the expansion of Green and Taylor. The Reed-Muller-Fourier form gives a compact representation for adders, while our expansion seems to be suitable for multipliers.

1. Introduction

Two-level expressions of multiple-valued logic functions and their minimization has been a subject of active research for many years. This problem is important because it provides a means for optimizing the implementations of circuits that are direct translations of two-level expressions ([1]-[4]). Thus, two-level logic representations have direct impact on macro-cell design styles using programmable logic arrays (PLAs).

We can classify two-level expressions of multiple-valued logic functions into two major groups. The first one includes logic representations evolved from classical sum-of-products form of Boolean functions over a Boolean algebra. An example is the canonical form of multiple-valued functions in a Post algebra over the set of operations (MIN, MAX, literal) [5]. The second group includes logic representations based on generalization of modulo 2 polynomial over a Galois Field $GF(2)$, often referred to as *Reed-Muller canonical form* [6], [7]. In this paper we consider three different generalizations of the Reed-Muller canonical form to multiple-valued logic and compare their complexity.

The Reed-Muller canonical form can be extended to multiple-valued logic in several ways, depending on how its operations are generalized. Many extensions have been suggested, including [8]-[13]. In these extensions, AND and XOR operations (which are equivalent to multiplication and addition modulo 2, correspondently) are generalized to addition and multiplication in $GF(m)$. This is a natural extension, allowing to preserve the nice properties of fields. However, such generalizations are only applicable to algebras with m being a prime or a power of a prime number. This doesn't cover, for example, the interesting case of $m = 10$. In [14] we introduced another generalization of the fixed polarity Reed-Muller canonical form, where AND is extended to MIN operation, and XOR is extended to addition modulo m. Since MIN is not distributive over addition modulo m, such an algebraic structure is not a field any longer, but it is applicable to any positive integer m.

From an implementation point of view, the MIN operation is easier to implement than the multiplication in $GF(m)$. For example, in current-mode CMOS technology, MIN can be implemented using 5 transistors only [15]. Moreover, this implementation is independent of m, i.e. the number of transistors does not increase with increasing values of m. On the other hand, the implementation of multiplication in $GF(m)$ is larger and it always depends on m. For example, for $m = 3$, a current-mode CMOS circuit realizing multiplication modulo 3 consists of 16 transistors

0-7695-0161-3/99 $10.00 © 1999 IEEE

[16].

However, this implementation advantage can only be utilized if the complexity of MIN-based expression of an m-valued function is not larger than the complexity of the $GF(m)$ multiplication-based expression. The purpose of this paper is to compare the complexity of our canonical form to the complexity of the canonical forms introduced in [11] and [13]. As a measure of complexity we use the number of products and the number of literals per term in the expression. This choice is motivated by the major optimization objectives in combinational logic design, which are *area* and *delay* reduction [17]. When considering a two-level implementation of a sum-of-products-type expression by a PLA, each row of a PLA is in one-to-one correspondence with a product-term of the sum-of-products expression. Each transistor in a row relates to the literals of a product-term. Therefore, the primary goal of logic optimization is the reduction of the number of products, and a secondary one is the reduction of the number of literals. Achieving a sum-of-products representation with the minimal number of products and with the minimal number of literals corresponds to optimizing both area and delay. So, these two parameters are good measures of the complexity of a two-level expression. For our purpose, a *minimal* expression is an expression with the smallest number of literals taken from the expressions with a smallest number of products.

Using the theory developed in [12], [13] and [14] we implemented an algorithm for computing the minimal canonical forms for the three different generalizations of Reed-Muller canonical form: the Galois Field-based expansion introduced by Green and Taylor [11], the Reed-Muller-Fourier form of Stanković and Moraga [13] and our generalization from [14]. We applied the algorithm to a set of encoded 4-valued benchmark functions, 3- and 4-valued adders and multipliers. The experimental results show that for the benchmark functions and multipliers Reed-Muller-Fourier form and our form have a comparable number of products on average. They have 40% less products on average than the expansion of Green and Taylor. Reed-Muller-Fourier form gives best results for adders, contrary to our form for which the adders seem to be the worst-case. However, our expansion seems to be suitable for multipliers.

The paper is organized as follows. In Section 2, the background on Reed-Muller canonical form and its generalizations is given. In Section 3, the algorithm for computing the minimal canonical forms from [11], [13] and [14] is described. In Section 4, the experimental results are shown. In the final section, some conclusions are drawn and directions for further research are proposed.

2. Reed-Muller canonical form and its generalizations

In this section we describe Reed-Muller canonical form of Boolean functions and its generalizations to multiple-valued logic.

Reed-Muller canonical form of Boolean functions was introduced in 1954 by Reed [6] and Muller [7]. It is an expression of type:

$$f(x_1,\ldots,x_n) = \sum_{i=0}^{2^n-1} c_i \cdot x_1^{i_1} \cdot x_2^{i_2} \cdot \ldots \cdot x_n^{i_n}, \quad (1)$$

where $c_i \in \{0,1\}$ are constants, the sign $\sum$ stands for XOR and "$\cdot$" stands for AND. $(i_1 i_2 \ldots i_n)$ is the binary expansion of i with i_1 being the least significant digit, and $x_j^0 = 1$ and $x_j^1 = x_j$ for $j \in \{0,1,\ldots,n\}$. All variables in (1) are uncomplemented.

If the restriction on all the variables being uncomplemented is dropped, then the Reed-Muller canonical form extends to *fixed polarity* Reed-Muller canonical form, which is unique for a fixed polarity $k \in \{0,1,\ldots,2^n-1\}$ and is given by:

$$f(x_1,\ldots,x_n) = \sum_{i=0}^{2^n-1} c_i \cdot {}^{k_1}x_1^{i_1} \cdot {}^{k_2}x_2^{i_2} \cdot \ldots \cdot {}^{k_n}x_n^{i_n} \quad (2)$$

where $(i_1 i_2 \ldots i_n)$ and $(k_1 k_2 \ldots k_n)$ are the binary expansions of i and k, respectively. The term ${}^{k_j}x_j^{i_j}, j \in \{0,1,\ldots,n\}$, is defined by: ${}^0x_j^1 = x_j$, ${}^1x_j^1 = x_j'$ and, ${}^{k_j}x_j^0 = 1$, for $k_j \in \{0,1\}$.

There are several ways to extend the Reed-Muller canonical form to m-valued logic. The first generalization was proposed by Cohn in 1960 [8]. It is an expression over a ring of integers modulo m, with m being a prime, namely:

$$f(x_1,\ldots,x_n) = \sum_{i=0}^{m^n-1} c_i \odot x_1^{i_1} \odot x_2^{i_2} \odot \ldots \odot x_n^{i_n}, \quad (3)$$

where c_i are constants over a finite set of values $M \stackrel{df}{=} \{0,1,\ldots,m-1\}$, the sign $\sum$ stands for addition modulo m, and "$\odot$" stands for multiplication modulo m. $(i_1 i_2 \ldots i_n)$ is the m-ary expansion of i, and the term $x_j^{i_j}$ denotes the i_jth power of the variable $x_j, j \in \{0,1,\ldots,n\}$.

Cohn's generalization was extended by Pradhan [9] for the case when m is a power of a prime, and then further extended by Kodandapani and Setlur [10] to the fixed-polarity case, where the variables x_j, $j \in \{1,2,\ldots,n\}$, are represented by all literals except $\overset{k_j}{x}_j$, where $(k_n \ldots k_2 k_1)$ is the

m-ary expansion of fixed a polarity $k \in \{0, 1, \ldots, m^n - 1\}$. Recall, that the literal operator $\overset{i}{x}$, $i \in M$, is defined by

$$\overset{i}{x} \overset{df}{=} \begin{cases} m-1 & \text{if } x = i \\ 0 & \text{otherwise} \end{cases}$$

Green and Taylor [11] introduced a different extension of Cohn's form to the fixed-polarity case, where an additive transform $x_j + k_j$ is performed on each variable x_j, $j \in \{1, 2, \ldots, n\}$, according to a fixed polarity $k \in \{0, 1, \ldots, m^n - 1\}$. Harking and Moraga [12] presented an efficient algorithm for computing such an expansion for different polarities.

Stanković and Moraga [13] introduced yet another extension, called *Reed-Muller-Fourier form*, where the restriction on all the variables being uncomplemented is dropped, and the NOT operation is generalized to be any reordering of the elements of $GF(m)$, giving $(m!)^n$ different fixed polarities.

All of the above described generalizations are only applicable for the algebras with m being a prime or a power of a prime number. In [14] we introduced another generalization of the fixed polarity Reed-Muller canonical form, applicable to any positive integer m. We showed that any m-valued n-variable function can be expressed over the set of operations addition modulo m, MIN and the set of all literal operators $\overset{i}{x}$, $i \in M$, in the following fixed-polarity canonical form:

$$f(x_1, \ldots, x_n) = \sum_{i=0}^{m^n-1} c_j \cdot {}^{k_1}x_1^{i_1} \cdot {}^{k_2}x_2^{i_2} \cdot \ldots \cdot {}^{k_n}x_n^{i_n} \quad (4)$$

where the sign $\sum$ stands for addition modulo m, and "$\cdot$" stands for MIN, and the term ${}^{k_j}x_j^{i_j}$ is defined by

$${}^{k_j}x_j^{i_j} \overset{df}{=} \begin{cases} m-1 & \text{if } k_j = 0 \\ \overset{k_j \oplus i_j}{x}_j & \text{otherwise} \end{cases}$$

where x_j, $j \in \{1, 2, \ldots, n\}$, is a multiple-valued variable and $k_j, i_j \in M$ are constants.

In our generalization of the polarity, we assume that each variable x_j, $j \in \{1, 2, \ldots, n\}$, is represented by all literals except $\overset{k_j}{x}_j$, where $(k_n \ldots k_2 k_1)$ is the m-ary expansion of a polarity $k \in \{0, 1, \ldots, m^n - 1\}$. For example, if $m = 3$, polarity vector $k = (021)$ implies that in the canonical form x_1 is represented by literals $\overset{0}{x}_1$ and $\overset{2}{x}_1$, x_2 is represented by $\overset{0}{x}_2$ and $\overset{1}{x}_2$, and x_3 is represented by $\overset{1}{x}_3$ and $\overset{2}{x}_3$. A similar generalization of polarity is used in [10].

We select three fixed polarity generalizations of Reed-Muller canonical form: the form introduced by Green and Taylor [11], the Reed-Muller-Fourier form of Stanković and Moraga [13] and our generalization from [14], and compare their complexity. In the next section we describe the algorithm which we use for computing these forms.

3. Computation of the minimal canonical forms

In this section we present an algorithm for computing the minimal canonical forms for the generalizations introduced in [11], [13] and [14]. The algorithm is based on the theory developed in [12], [13], [14], and, apart from unifying the previous results, brings no novelty. The reason for reporting it here is merely to give the reader a clear picture of how the experiment was conducted, so that it can be repeated, if needed.

The pseudocode of the algorithm is shown in Figure 2. The same structure is used for computing all three canonical forms, with the difference in the transformation matrices and the operations used for the computations. For example, the transformation matrices for $n = 1$ and $m = 4$ are shown in Figure 1.

Green & Taylor	Stanković & Moraga	Dubrova & Muzio
$\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 3 & 2 & 1 \\ 0 & 2 & 3 & 1 \end{bmatrix}$	$\begin{bmatrix} 3 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 \\ 3 & 2 & 3 & 0 \\ 3 & 3 & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 \\ 3 & 0 & 1 & 0 \\ 3 & 0 & 0 & 1 \end{bmatrix}$

Figure 1. The transformation matrices for $n = 1$ and $m = 4$.

The algorithm receives as an input a cube cover of the on-set of a completely specified multiple-valued multiple-output function f. It returns as an output the number of non-zero coefficients in a minimal canonical form of f. The number of non-zero coefficients corresponds to the number of products in the expression. Recall, that we define *minimal* expression as the expression with the smallest number of literals among the expressions with a smallest number of products. An *optimal* polarity is the polarity of a minimal expression.

First, the transformation matrix $TM(n)$ is generated. $TM(n)$ is a two-dimensional array of integers of size $m^n \times m^n$, where m is the radix of logic and n is the number of inputs.

The arrays P and R are two-dimensional arrays of integers of size $m^n \times k$, where k is the number of outputs of f. The polarity vector P is computed for a given polarity p by reordering the coefficients of the functions in a certain way, depending of the value of p (for more details the reader is referred to [12], [13] and [14]). R is the resulting vector of coefficients of the canonical form. It is obtained by multiplying the transformation matrix TM by the polarity vector P. For Reed-Muller-Fourier form [13] and our form [14],

the computation is carried out modulo m. For the Green and Taylor form [11], the operations of multiplication and addition over $GF(m)$ are used. For a prime m these coincide with modulo m operations. For the extended fields, when m is a power of a prime, they differ.

Notice, that the algorithm processes multiple-output functions by first fixing the polarity, then computing the non-zero coefficients of the expressions for all outputs, and then evaluating the total number of products. We keep track of the products, common for several outputs, are count them just once. Another possible strategy would be to find an optimal polarity for each output function separately.

GeneralizedRM(F)
input: on-set F of f
output: number of non-zero coefficients in a minimal canonical form of f

```
TM(n) ← Generate(n);
best_products = ∞;
best_literals = ∞;
best_polarity = 0;
for (all polarities p) {
        make the polarity vector P by reordering the
        coefficients of f;
        R = TM(n) · P;
    products ← CountProducts(R);
    literals ← CountLiterals(R);
    if (products ≤ best_products) {
        if (literals < best_literals) {
            best_products = products;
            best_literals = literals;
            best_polarity = p;
        }
    }
}
re-compute R and P for best_polarity;
TM^-1(n) ← GenerateReverse(n);
if (TM^-1(n) · R ≠ P)
    return("error");
else
    return(best_products);
```

Figure 2. Pseudocode of the algorithm for computing minimal canonical forms.

After the optimal polarity is found, the solution is verified by performing the reverse transform $TM^{-1}(n) \cdot R$.

4. Experimental results

We implemented the algorithm shown in Figure 2 and applied it to a set of 4-valued benchmark functions, and 3- and 4-valued adders and multipliers. The m-valued adders and multipliers were obtained by a direct encoding of their functionality in m-valued logic. The 4-valued benchmark functions[1] were generated by pairing inputs *and* outputs of benchmarks and encoding them using the scheme shown in Figure 3. If a function has an odd number of inputs (or outputs), then we assume the existence of one additional input (or output) which has the don't care value for all combinations. Since our algorithm takes as an input completely specified functions only, we first run Espresso [18] to obtain covers of the original benchmarks and then apply the encoding to the resulting covers. However, a similar strategy can be used to obtain a set of incompletely specified 4-valued functions. The fragments of Espresso format of a Boolean function and the resulting 4-valued one are shown in Figure 4. Notice, that if a single value in a pair of binary values is don't care, then this pair is encoded in two different ways. A binary cube with r such pairs will be encoded in 2^r 4-valued cubes.

pair of values in the original function	resulting value(s) in the 4-valued function
00	0
01	1
10	2
11	3
--	-
-0	{0,2}
-1	{1,3}
0-	{0,1}
1-	{2,3}

Figure 3. Encoding scheme used to obtain 4-valued input 4-valued output benchmarks; "-" stands for the don't care value.

Table 1 shows the number of non-zero coefficients in the expressions computed by the algorithm in Figure 2 for the 4-valued benchmark functions. Columns 2 and 3 give the number of inputs n and the outputs k of the functions. The numbers of non-zero coefficients in the canonical forms [11], [13] and [14] are shown in the columns 4, 6 and 8, correspondently. To have a more representative comparison, we have also presented in columns 5, 7 and 9 the relative values in %, taking the Green & Taylor [11] form as ba-

[1]This set of 4-valued benchmarks is available at http://www.ele.kth.se/~elena/

Table 1. Benchmark results for 4-valued functions, showing the number of non-zero coefficients in the expressions computed by the algorithm in Figure 2.

Example function	n	k	Green & Taylor [11]		Stanković & Moraga [13]		Dubrova & Muzio [14]	
			abs.	%	abs.	%	abs.	%
5xp1	3	5	223	100%	154	69.06%	150	67.26%
9sym	5	1	261	100%	84	32.18%	135	51.72%
bw	3	14	118	100%	79	66.95%	96	81.36%
clip	5	3	1112	100%	673	60.52%	715	64.30%
con	4	1	57	100%	42	73.68%	23	40.35%
ex1010	5	5	2249	100%	2231	99.20%	2238	99.51%
inc	4	5	193	100%	123	63.73%	120	62.18%
misex1	4	4	99	100%	51	51.52%	28	28.28%
rd53	3	2	50	100%	25	50.00%	30	60.00%
rd73	4	2	189	100%	106	56.08%	120	63.49%
rd84	4	2	262	100%	125	47.71%	116	44.27%
sao2	5	2	657	100%	246	37.44%	89	13.55%
squar5	3	4	61	100%	38	62.30%	47	77.05%
average	4	4	425	100%	306	59.26%	301	57.95%

Table 2. Experimental results for 3- and 4-valued adders, showing the number of non-zero coefficients in the expressions computed by the algorithm in Figure 2.

m	Example function	n	k	Green & Taylor [11]		Stanković & Moraga [13]		Dubrova & Muzio [14]	
				abs.	%	abs.	%	abs.	%
3	adder 2-bit	4	3	21	100%	14	66.67%	32	152.38%
	adder 3-bit	6	4	75	100%	42	56.00%	165	220.00%
4	adder 2-bit	4	3	77	100%	34	44.16%	68	88.31%
	adder 3-bit	6	4	415	100%	174	41.93%	524	126.27%
	average	5	4	147	100%	66	52.19%	197	146.74%

Table 3. Experimental results for 3- and 4-valued multipliers, showing the number of non-zero coefficients in the expressions computed by the algorithm in Figure 2.

m	Example function	n	k	Green & Taylor [11]		Stanković & Moraga [13]		Dubrova & Muzio [14]	
				abs.	%	abs.	%	abs.	%
3	mult. 2-bit	4	4	52	100%	33	63.46%	31	59.62%
	mult. 3-bit	6	6	581	100%	369	63.51%	349	60.07%
4	mult. 2-bit	4	4	278	100%	113	40.36%	108	38.85%
	mult. 3-bit	6	6	5813	100%	2830	48.68%	2171	37.35%
	average	5	4	1681	100%	836	54.08%	665	48.97%

```
.i 9                          .i 5
.o 10                         .o 5
                              .m 4
110000-10  1000100001         30010  20201
                              30030  20201
                              30011  20201
                              30031  20201
...                           ..
10--10011  0000100100         2-212  00210
                              2-213  00210
.e                            .e
```

Figure 4. Fragment of Espresso format of an original benchmark function and the encoded 4-valued one.

sis (100%). Tables 2 and 3 show the results for the 3- and 4-valued adders and multipliers, correspondently.

The experimental results show that for the benchmark functions Reed-Muller-Fourier form and our expansion are comparable in the number of products on average. The expansion of Green and Taylor has 40% more products on average. Reed-Muller-Fourier form gives best results for adders, contrary to our form for which the adders are hard. Our form seems particularity suitable for multipliers, with 5% reduction of the number of products over Reed-Muller-Fourier form and 51% reduction over Green and Taylor's form.

For the Green and Taylor's and our expansions, the optimal polarity is computed by an exhaustive search through all m^n possible polarities. This brute force approach makes the algorithm infeasible for the functions with $n > 6$ and $m > 4$. E.g., it takes only 4 min to compute the optimal polarity for the 3-valued 3-bit adder having 6 inputs. However, 2 days are needed to find the solution for the 3-valued 4-bit adder having 8 inputs. Using some heuristics for finding an optimal polarity would extend the capability of the algorithm, but so far no such heuristics have been found.

For Reed-Muller-Fourier form [13], the exhaustive search through all $(m!)^n$ possible polarities is out of question even for $n > 2$. Therefore, we compute m^n representatives of $(m!)^n$ possible polarities and take the best result. This, of course, means that we do not exploit the full power of Reed-Muller-Fourier form and for some functions the result we compute is not the best one.

When analyzing the results in the above tables, it should be taken into account that the products in the Reed-Muller-Fourier form and the form [11] use $GF(m)$ multiplication, while the products in our form [14] use MIN. With present technologies, MIN has a much simpler implementation than multiplication in $GF(m)$. So, the circuit realizing a MIN-based expression will occupy less area than the circuit realizing a $GF(m)$ multiplication-based expression of the same number of products and literals. This, of course, might change as technology evolves.

5. Conclusion

This paper compares the complexity of three fixed polarity generalizations of Reed-Muller canonical form: the Galois Field-based form introduced by Green and Taylor [11], the Reed-Muller-Fourier form of Stanković and Moraga [13] and our generalization over addition modulo m, minimum and the set of all literal operators [14]. Using the theory developed [12], [13] and [14] we implemented an algorithm for computing the minimal canonical forms for the corresponding generalizations and applied to a set of benchmark functions, adders and multipliers. The experimental results show that for the benchmark functions the Reed-Muller-Fourier form and our expansion give comparable results. On average, the number of products in these expansions is 40% smaller than the number of products in the expansion of Green and Taylor. Adders have most compact representations in Reed-Muller-Fourier form. Our expansion seems to be most suitable for multipliers.

In general, it is hard to make a comparison between different representations of functions. We based our comparison on the benchmarks results. An alternative would be to compare the upper bounds on the number of products in the canonical forms. Unfortunately, very few results on the upper bounds are reported for the case of $m > 2$ and none for the canonical forms introduced in [11], [13] and [14].

The main open problem remains finding a good heuristic for selecting an optimal polarity. We need to examine which functional properties can be used to guide the choice of polarity and also to investigate which search techniques are applicable to this problem.

Further work on the efficiency of computing the expressions might also incorporate representation of multiple-valued functions by Multiple-Valued Decision Diagrams (MDD) [19], and performing the basic operations of the algorithm directly on graphs. Since there is no direct correspondence between the size of the cover for the function and the size of the MDD that represents it, very large cube covers can be captured and efficiently manipulated using this representation. For the Reed-Muller-Fourier representation, some work in this direction for has been started in [20] and [21].

Acknowledgment

I am grateful to Prof. Claudio Moraga and Prof. Radomir Stanković for helpful discussions over the Reed-Muller-

Fourier form and to Dr. Dilian Gurov for carefully reading and commenting on the manuscript. I am also indebted to the anonymous referee who gave me the reference [11].

This work was supported in part by Research Grant No 240-98-101 from the Swedish Research Council for Engineering Sciences and by a fellowship from the Knut and Alice Wallenbergs foundation of Sweden.

References

[1] T. Sasao, On the optimal design of multiple-valued PLA's, *IEEE Trans. Comput.* **C-38** No. 4 (1989), 582-592.

[2] M. A-E-Barr, M. N. Hasen, New MVL-PLA structure based on current-mode CMOS technology, *Proc. 26th Int. Symp. Multiple-Valued Logic*, (1996), 98-103.

[3] X. Deng, T. Hanyu, M. Kameyama, Design and evaluation of a current-mode multiple-valued PLA based on a resonant tunneling transistor model, *IEE Proc. Circuits Devices Syst.* **141** No. 6 (1994), 445-450.

[4] T. Utsumi, N. Kamiura, Y. Hata, K. Yamato, Multiple-valued programmable logic array with universal literals, *Proc. 27th Int. Symp. Multiple-Valued Logic*, (1997), 163-169.

[5] J.C. Muzio, T.C. Wesselkamper, *Multiple-Valued Switching Theory*, Adam Hilger Ltd. Bristol and Boston, 1986.

[6] I. S. Reed, A class of multiple-error-correcting codes and their decoding scheme, *IRE Trans. on Inform. Theory* **4** (1954), 38-42.

[7] D. E. Muller, Application of Boolean algebra to switching circuit design and to error detection, *IRE Trans. Electron. Comput.* **EC-3** (1954), 6-12.

[8] M. Cohn, *Switching Function Canonical Form over Integer Fields*, Ph.D thesis, Harvard University, Cambridge, Mass., Dec. 1960.

[9] D. K. Pradhan, A multi-valued algebra based on finite fields, *Proc. 1974 International Symp. on MVL*, (1974) 95-112.

[10] K. L. Kodandapani, R. V. Setur, Multi-valued algebraic generalization of Reed-Muller canonical forms, *Proc. 1974 International Symp. on MVL* (1974), 505-544.

[11] D. H. Green, I. S. Taylor, Modular representation of multiple-valued logic systems *Proc. of the IEE*, **121**, (1974), 424-429.

[12] B. Harking, C. Moraga, Efficient derivation of Reed-Muller expansions in multiple-valued logic systems, *Proc. 22nd International Symp. on MVL* (1992), 436-441.

[13] R. S. Stanković, C. Moraga, Reed-Muller-Fourier versus Galois Field representations of four-valued logic functions, *Proc. 28th International Symp. on MVL* (1998), 186-191.

[14] E. V. Dubrova, J. C. Muzio, Generalized Reed-Muller canonical form for a multiple-valued algebra, *Multiple-Valued Logic, An International Journal* **1** (1996), 65-84.

[15] L. Zhijian, J. Hong, A CMOS current-mode high speed fuzzy logic microprocessor for a real-time expert system, *Proc. 20th International Symp. on MVL* (1990), 394-400.

[16] Z. Zilic, Z. Vranesic, Current-mode CMOS Galois field circuits, *Proc. 23rd International Symp. on MVL* (1993), 245-250.

[17] G. De Micheli, *Synthesis and optimization of digital circuits*, McGraw-Hill, 1994.

[18] R.K. Brayton, G. Hachtel, C. McMullen, A. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publisher, 1984.

[19] D. M. Miller, Multiple-valued logic design tools, *Proc. 22rd International Symp. on MVL* (1993), 2-11.

[20] R. S. Stanković, C. Moraga, T. Sasao, Calculation of Reed-Muller-Fourier coefficient of multiple-valued functions through multiple-place decision diagrams, *Proc. 24th International Symp. on MVL* (1994), 82-88.

[21] R. S. Stanković, Functional decision diagrams for multiple-valued functions, *Proc. 25th International Symp. on MVL* (1995), 284-289.

Multiple-Valued Minimization to Optimize PLAs with Output EXOR Gates

Debatosh Debnath and Tsutomu Sasao
Department of Computer Science and Electronics
Kyushu Institute of Technology
Iizuka 820-8502, Japan

Abstract

This paper considers an optimization method of programmable logic arrays (PLAs), which have two-input EXOR gate at the outputs. The PLA realizes an EXOR of two sum-of-products expressions (EX-SOP) for multiple-valued input two-valued output functions. We present techniques to minimize EX-SOPs, which is an extension of Dubrova-Miller-Muzio's AOXMIN algorithm. We conjecture that, when n is sufficiently large, an EX-SOP for n-bit adder requires at most 2^n products while an ordinary sum-of-products expression (SOP) requires $6 \cdot 2^n - 4n - 5$ products. Experimental results for two- and four-valued benchmark functions show that the proposed method produces better EX-SOPs than existing methods.

Index Terms—Three-level network, logic minimization, adder, multiple-valued logic, programmable logic array.

1 Introduction

Programmable logic arrays (PLAs) with two-input EXOR gate at the outputs, also known as AND-OR-EXOR PLAs (Fig. 1), are a powerful architecture to realize many logic functions. The AND-OR-EXOR PLA realizes an EXOR of two sum-of-products expressions (EX-SOP). Minimization of the number of products in EX-SOPs is an important step in the optimization of AND-OR-EXOR PLAs, because the number of products is directly related to the cost of PLAs. EX-SOPs are promising because, for many practical logic functions, they often require many fewer products than sum-of-products expressions (SOPs) [4, 6, 13].

Minimization of EX-SOPs were considered in the past [8, 14] and a cut-and-try method was reported [9]. Design methods for adders by using AND-OR-EXOR PLAs with more than one-bit decoders were developed at IBM [15]. In the last few years significant progress in the minimization of EX-SOPs have been made [4, 6, 13]. Upper bounds on the number of products in EX-SOPs are reported [2, 3, 5]. AND-OR-EXOR network where the output EXOR gate have unlimited fan-in is considered [12].

In this paper, we present a heuristic method to minimize EX-SOPs, which is an extension of AOXMIN [6]. Unlike AOXMIN, we can minimize EX-SOPs for multiple-valued input two-valued output functions. EX-SOPs for functions with two- and four-valued inputs correspond to AND-OR-EXOR PLAs with one- and two-bit decoders, respectively (Fig. 1) [13]. We also present a method to further reduce the number of products in EX-SOPs by considering output phase optimization [11], where some components of the function are implemented in the complemented form.

A crucial step in AOXMIN is to partition the products of an SOP of the given function into two sets, which is done by a random method. We propose a partitioning method for adders. Our experimental result demonstrates that, for an n-bit adder with two-valued inputs and sufficiently large n, the proposed partitioning method is about 250 times faster to produce about two times better solution than the random partitioning method. For adders with two-bit decoders, proposed partitioning method is faster than random partitioning method in producing comparable solutions.

The remainder of the paper is organized as follows: Section 2 reviews terminologies. Section 3 considers output phase optimization techniques. Section 4 summarizes AOXMIN and describes its extensions. Section 5 presents design method for adders. Section 6 shows experimental results and conjectures that, when n is sufficiently large, an EX-SOP for an n-bit adder requires at most 2^n products. Section 7 presents conclusions.

2 Definitions and Terminologies

In this section, we review basic terminologies related to multiple-valued functions [10, 11].

Definition 2.1 *A **multiple-valued input two-valued output function**, or **function** in short, is a mapping*

$$f(X_1, X_2, \ldots, X_n) : \bigtimes_{i=1}^{i=n} P_i \rightarrow B,$$

*where $P_i = \{0,1,\ldots,p_i - 1\}$, $p_i \geq 2$, $B = \{0,1\}$, and X_i is a **multiple-valued variable** taking a value from P_i.*

Definition 2.2 *Let $S_i \subseteq P_i$. A **literal** $X_i^{S_i}$ represents 0 if $X_i \notin S_i$ and 1 if $X_i \in S_i$. A **product** $X_1^{S_1} X_2^{S_2} \cdots X_n^{S_n}$ is AND of literals. A **cube** is a convenient representation of a product for computer manipulation.*

0-7695-0161-3/99 $10.00 © 1999 IEEE

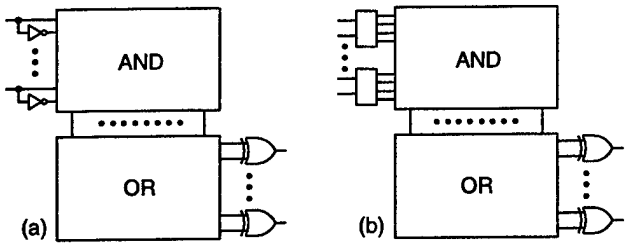


Figure 1: AND-OR-EXOR PLA with (a) one-bit and (b) two-bit decoders.

Definition 2.3 *A **sum-of-products expression (SOP)***

$$\bigvee_{(S_1,S_2,\ldots,S_n)} X_1^{S_1} X_2^{S_2} \cdots X_n^{S_n}$$

*is OR of products. An SOP is represented by a **cover**, which is a set of cubes. An **EX-SOP***

$$\bigvee_{(S_1,S_2,\ldots,S_n)} X_1^{S_1} X_2^{S_2} \cdots X_n^{S_n} \oplus \bigvee_{(S_1,S_2,\ldots,S_n)} X_1^{S_1} X_2^{S_2} \cdots X_n^{S_n} \qquad (2.1)$$

is the EXOR of two SOPs.

Definition 2.4 *Let $f_i(X_1,X_2,\ldots,X_n)$ $(i = 0,1,\ldots,m-1)$ be an n-input m-output function. Then, the two-valued output function $F(X_1,X_2,\ldots,X_n,X_{n+1})$, where X_{n+1} is an m-valued variable representing the outputs such that $F(X_1,X_2,\ldots,X_n,i) = f_i(X_1,X_2,\ldots,X_n)$, is the **characteristic function** for the multiple-output function [11].*

Definition 2.5 *An **SOP for a multiple-output function** indicates an SOP for its characteristic function, and an **EX-SOP for a multiple-output function** indicates an EX-SOP for its characteristic function.*

Definition 2.6 *The **intersection** of the products $c_1 = X_1^{S_1} X_2^{S_2} \cdots X_n^{S_n}$ and $c_2 = X_1^{T_1} X_2^{T_2} \cdots X_n^{T_n}$, denoted by $c_1 \cap c_2$, is the product $X_1^{S_1 \cap T_1} X_2^{S_2 \cap T_2} \cdots X_n^{S_n \cap T_n}$. If $S_i \cap T_i = \emptyset$ for some i, then the intersection denotes a null cube.*

Definition 2.7 *Disjoint sharp of two covers F and G, denoted by $F \textcircled{\#} G$, represents only those minterms of F which are not contained by G.*

Definition 2.8 *ON-set, OFF-set, and DC-set is the set of cubes for which the function value is 1, 0, and unspecified, respectively.*

In this paper, we often use the same symbol for a function and its cover; and unless otherwise specified, *adder* refers to *adder without carry input*, and adrn represents an n-bit adder.

3 Output Phase Optimization

In many cases, we can realize a function f in either *positive phase* (f) or *negative phase* ($\bar{f}$). For m-output function, we can choose the output phases in 2^m ways. The choice of the output phases in the realization of a function influences on the number of products in its minimized expressions. To reduce the number of products by choosing the output phases is *output phase optimization* [11].

Definition 3.1 *Let $(f_0,f_1,\ldots,f_{m-1})$ be an m-output function. Then, the minimized SOP G for the characteristic function of $(g_0,g_1,\ldots,g_{m-1})$, where $g_i \in \{\bar{f}_i,f_i\}$ $(i = 0,1,\ldots,m-1)$ such that the number of products in G is minimal, is the **output phase optimized SOP** for $(f_0,f_1,\ldots,f_{m-1})$.*

Similarly, we can define an *output phase optimized EX-SOP*.

We handle the output phase optimization of EX-SOPs by using the output phase optimization of SOPs. We use an output phase optimized SOP as the input of the EX-SOP minimizer. For a function with m outputs, an EX-SOP minimizer produces two SOPs each having m outputs. We optimize the output phases of the $2m$-output SOP to obtain an output phase optimized EX-SOP.

Let the output phase for the function f_i be $a_i \in \{0,1\}$, where $a_i = 0$ indicates f_i is in the positive phase and $a_i = 1$ indicates f_i is in the negative phase. Let the output phases of the two SOPs of the EX-SOP for f_i be b_{i0} and b_{i1}. Therefore, the output phase of the EX-SOP for f_i is $a_i \oplus b_{i0} \oplus b_{i1}$. When output phase optimization of the two m-output SOPs is impractical, we consider a_i as the output phase of the EX-SOP for f_i.

An output phase optimized EX-SOP can be realized in an AND-OR-EXOR PLA, where the polarity of the outputs are programmable.

4 Simplification Techniques

In this section, we review AOXMIN [6], which is a heuristic algorithm to simplify EX-SOPs. We then present an extension of AOXMIN.

4.1 An Overview of AOXMIN [6]

Basic steps of AOXMIN are as follows:

1. Obtain a minimized cover F for the given function f and compute a cover R for $\bar{f}$.
2. Group the cubes of F into clusters of cubes. Two cubes are in the same *cluster* if they intersect or they are connected through a chain of intersecting cubes. (In [6], a cluster of cubes are called an equivalence class.)
3. Randomly partition the cluster of cubes into two covers, F_A and F_B.
4. Obtain two EX-SOPs by using AOXMIN_SPECIFY-_BOTH(F_A,F_B,R) and AOXMIN_SPECIFY_BOTH(F_B,F_A,R) (Fig. 2). AOXMIN_SPECIFY_BOTH returns two SOPs which form an EX-SOP. ESPRESSO(F_k,D_k,R_k) in Fig. 2 obtains a minimized cover for a function, where F_k, D_k, and R_k represents the ON-set, DC-set, and OFF-set, respectively.
5. Iterate steps 3 and 4 for some specified number of times, and take the best EX-SOP among all the EX-SOPs generated so far.

In addition, AOXMIN simplifies complement of the given function and uses some output phase optimization technique to obtain better solution.

```
1 procedure AOXMIN_SPECIFY_BOTH(F_A,F_B,R) {
2     F_a ← ESPRESSO(F_A,R,F_B);
3     R_assigned ← F_a ⊕ F_A;
4     F_temp ← F_B ∪ R_assigned;
5     R_temp ← F_A ∪ (R ⊕ R_assigned);
6     F_b ← ESPRESSO(F_temp,∅,R_temp);
7     return (F_a,F_b);
8 }
```

Figure 2: Pseudocode AOXMIN_SPECIFY_BOTH.

4.2 An Extension of AOXMIN

The proposed heuristic method to simplify EX-SOPs, which is an extension of AOXMIN [6], have the following features:

- It can simplify EX-SOPs for functions with two- and four-valued variables, and can treat functions where different variables have different values. On the other hand, AOXMIN simplifies only two-valued functions.
- It uses heuristic algorithms to partition the cluster of cubes for adders. In this regard, AOXMIN uses only a random partitioning method.
- During iterative improvement, it concurrently minimizes both SOPs of the EX-SOP to reduce the total number of products by increasing shared products between two SOPs. On the other hand, AOXMIN uses simultaneous minimization of both SOPs only once as part of its simplification technique for multiple-output functions.
- For multiple-output functions, it performs concurrent simplification of all the outputs. However, AOXMIN simplifies each output separately throughout the algorithm. A modified AOXMIN considers simplification of all the outputs simultaneously [7].
- For the output phase optimization of EX-SOPs, it uses techniques for the output phase optimization of SOPs [11]. AOXMIN handles the output phase optimization problem in a different way.
- To find good solutions quickly, especially for adders, it selects from two different minimizers for SOPs. On the other hand, AOXMIN uses only Espresso [1].
- The method makes efficient use of the given don't care conditions during grouping the cover into cluster of cubes and also during every minimization of the SOPs of the EX-SOP. AOXMIN does not use don't care conditions during these two operations.

Theorem 4.1 *An arbitrary multiple-valued input two-valued output function can be represented by an EX-SOP of the form (2.1).*

The minimization of SOP for a multiple-output function corresponds to the minimization of SOP for its characteristic function [11]. Similarly, we can prove the following:

```
1 procedure MODIFIED_SPECIFY_BOTH(F_A,F_B,D,R) {
2     F_AsharpD ← F_A ⊕ D;
3     F_BsharpD ← F_B ⊕ D;
4     F_a ← SIMPLIFY_SINGLE(F_AsharpD,D ∪ R,F_BsharpD);
5     R_assigned ← F_a ∩ R;
6     R_remained ← R ⊕ R_assigned;
7     F_b ← F_BsharpD ∪ R_assigned;
8     R_a ← F_BsharpD ∪ R_remained;
9     R_b ← F_AsharpD ∪ R_remained;
10    F_dbl ← MAKE_DOUBLE_OUT_COVER(F_a,F_b);
11    R_dbl ← MAKE_DOUBLE_OUT_COVER(R_a,R_b);
12    D_dbl ← MAKE_DOUBLE_OUT_COVER(D,D);
13    F_EX-SOP ← SIMPLIFY_DOUBLE(F_dbl,D_dbl,R_dbl);
14    return F_EX-SOP;
15 }
```

Figure 3: Pseudocode MODIFIED_SPECIFY_BOTH.

```
    /* F = ON-set, D = DC-set, R = OFF-set */
1 procedure SIMPLIFY_LOCAL(F,D,R) {
2     F ← REDUCE(F,D);
3     F ← EXPAND(F,R);
4     F ← IRREDUNDANT(F,D);
5     return F;
6 }
```

Figure 4: Pseudocode SIMPLIFY_LOCAL.

Theorem 4.2 *The minimization of EX-SOP for a multiple-output function corresponds to the minimization of EX-SOP for its characteristic function.*

Now, the definition of the cluster of cubes can be extended as follows:

Definition 4.1 *Let F and D be the covers for the ON-set and DC-set, respectively, of the characteristic function for a multiple-output function. Then, two cubes $c_i,c_j \in F$ are in the same cluster if*

(a) $G(i,j) \neq \emptyset$, *or*

(b) $G(i,i+1) \neq \emptyset, G(i+1,i+2) \neq \emptyset, \ldots, G(j-1,j) \neq \emptyset$,

where $G(p,q)$ *denotes* $(c_p \cap c_q) \oplus D$.

Section 4.1 shows that during every iteration AOXMIN calls AOXMIN_SPECIFY_BOTH twice. We replaced these calls by MODIFIED_SPECIFY_BOTH(F_A,F_B,D,R) and MODIFIED_SPECIFY_BOTH(F_B,F_A,D,R) (Fig. 3). MAKE_DOUBLE-_OUT_COVER(F_k,G_k) in Fig. 3 receives n-input m-output covers F_k and G_k, and returns an n-input $2m$-output cover such that covers corresponding to outputs $0,1,\ldots,m-1$ and $m,m+1,\ldots,2m-1$ represent F_k and G_k, respectively.

In Fig. 3, both SIMPLIFY_SINGLE(F_k,D_k,R_k) and SIMPLIFY_DOUBLE(F_k,D_k,R_k) obtain a minimized cover for a function, where F_k, D_k, and R_k represents the ON-set,

adr3: $1_5, 2_2, 3_2, 5_2$
adr4: $1_5, 2_2, 3_2, 5_2, 7_2, 11_2$
adr5: $1_5, 2_2, 3_2, 5_2, 7_2, 11_2, 15_2, 23_2$
adr6: $1_5, 2_2, 3_2, 5_2, 7_2, 11_2, 15_2, 23_2, 31_2, 47_2$
adr7: $1_5, 2_2, 3_2, 5_2, 7_2, 11_2, 15_2, 23_2, 31_2, 47_2, 63_2, 95_2$
adr8: $1_5, 2_2, 3_2, 5_2, 7_2, 11_2, 15_2, 23_2, 31_2, 47_2, 63_2, 95_2, 127_2, 191_2$
adr9: $1_5, 2_2, 3_2, 5_2, 7_2, 11_2, 15_2, 23_2, 31_2, 47_2, 63_2, 95_2, 127_2, 191_2, 255_2, 383_2$
adr10: $1_5, 2_2, 3_2, 5_2, 7_2, 11_2, 15_2, 23_2, 31_2, 47_2, 63_2, 95_2, 127_2, 191_2, 255_2, 383_2, 511_2, 767_2$
adr11: $1_5, 2_2, 3_2, 5_2, 7_2, 11_2, 15_2, 23_2, 31_2, 47_2, 63_2, 95_2, 127_2, 191_2, 255_2, 383_2, 511_2, 767_2, 1023_2, 1535_2$

Figure 5: Distribution of the clusters of output phase optimized SOPs for adders with two-valued inputs.

DC-set, and OFF-set, respectively. SIMPLIFY_SINGLE and SIMPLIFY_DOUBLE can be either SIMPLIFY_LOCAL (Fig. 4) or Espresso-MV [10]. SIMPLIFY_LOCAL uses a single pass of REDUCE, EXPAND, and IRREDUNDANT operations to obtain a simplified SOP [10]. It reduces the number of cubes by locally changing the shape of the cubes. Espresso-MV iterates these operations as long as the solution improves. Sections 5 and 6 explain how the choice of the two-level minimizers affect the quality of the solution and execution time.

5 Design of Adders

In this section, we propose partitioning methods of the cluster of cubes for adders with one- and two-bit decoders, and discuss about the choice of the two-level minimizers. Note that EX-SOPs for functions with two- and four-valued inputs correspond to AND-OR-EXOR PLAs with one- and two-bit decoders, respectively (Fig. 1).

During minimization of adders, we use SIMPLIFY_LOCAL for SIMPLIFY_SINGLE and Espresso-MV for SIMPLIFY_DOUBLE in Fig. 3. We observe that if Espresso-MV is used for SIMPLIFY_SINGLE then the resulting awkward shape of $R_{assigned}$ in Fig. 3 prevent us from obtaining a good solution in the next minimization by using SIMPLIFY_DOUBLE.

5.1 Adders with One-Bit Decoders

We found that output phase optimized SOP for n-bit ($3 \leq n \leq 11$) adder with two-valued inputs have $4n - 1$ cluster of cubes. Fig. 5 shows the distribution of these clusters, where an entry c_k represents k clusters each having c cubes. It is interesting that the number of cubes in the clusters have a regular structure. To partition the cluster of cubes into two covers F_A and F_B, we use the following method:

1. Sort the clusters in descending order of the number of cubes in it.
2. Starting from the beginning of the sorted list of the clusters, alternatively add a pair of clusters to F_A and a pair of clusters to F_B.
3. Add the remaining cluster to F_B.

Example 5.1 *For three-bit adder with two-valued inputs, the number of cubes in the clusters which form F_A and F_B are 5, 5, 2, 2, 1, 1, and 3, 3, 1, 1, 1, respectively.*

adr4: $1_4, 2_2, 3_2$
adr5: $1_4, 2_2, 3_2, 4_2$
adr6: $1_4, 2_2, 3_2, 4_2, 5_2$
adr7: $1_4, 2_2, 3_2, 4_2, 5_2, 6_2$
adr8: $1_4, 2_2, 3_2, 4_2, 5_2, 6_2, 7_2$
adr9: $1_4, 2_2, 3_2, 4_2, 5_2, 6_2, 7_2, 8_2$
adr10: $1_4, 2_2, 3_2, 4_2, 5_2, 6_2, 7_2, 8_2, 9_2$
adr11: $1_4, 2_2, 3_2, 4_2, 5_2, 6_2, 7_2, 8_2, 9_2, 10_2$

Figure 6: Distribution of the clusters of output phase optimized SOPs for adders with two-bit decoders.

The above partitioning method is devised by considering outputs. Adders have pairs of clusters, where each pair belongs to a particular set of outputs. Roughly, the strategy is to put the clusters from such pair into two different partitions. A similar method is also devised for adders with four-valued inputs.

5.2 Adders with Two-Bit Decoders

We obtained functions with four-valued inputs from their two-valued counterparts by pairing two variables using Espresso-MV [10]. Fig. 6 shows the distribution of the clusters of output phase optimized SOPs for adders with two-bit decoders, where an entry c_k represents k clusters each having c cubes. It shows that the output phase optimized SOP for n-bit ($4 \leq n \leq 11$) adder with two-bit decoders have $2n$ clusters. Note that the number of cubes in the clusters for adders with two-bit decoders also have a regular structure. We use the following method to partition the clusters into two covers F_A and F_B:

1. Sort the clusters in descending order of the number of cubes in it.
2. Starting from the beginning of the sorted list of the clusters, at first add a pair of clusters to F_A, then alternatively add a cluster to F_A and F_B.

6 Experimental Results

We implemented the proposed heuristic method to simplify EX-SOPs, which is an extension of AOXMIN [6], in C by using Espresso-MV [10] routines. On an HP C160 workstation with 256 megabytes memory resources, we conducted

Table 1: Experimental result for adders with two-valued inputs.

					Proposed Partition			Random Partition 20 Iterations		Random Partition 50 Iterations	
Data	In	Out	SOP	OPO SOP	EX-SOP	Time	OPO EX-SOP	EX-SOP	Time	EX-SOP	Time
adr3	6	4	31	25	12	0.02	11	17	1.24	13	2.95
adr4	8	5	75	61	21	0.07	18	32	4.48	32	13.46
adr5	10	6	167	137	37	0.35	36	50	26.04	50	61.98
adr6	12	7	355	293	67	1.45	66	146	114.38	133	332.95
adr7	14	8	735	609	122	4.56	120	128	422.94	128	1033.24
adr8	16	9	1499	1245	233	19.58	MEM	423	1634.73	380	3972.45
adr9	18	10	3031	2521	454	66.42	MEM	840	6469.34	840	16492.60
adr10	20	11	6099	5077	967	312.17	MEM	2168	28767.08	1898	74434.15
adr11	22	12	12239	10193	1993	1596.42	MEM	4136	169809.86	3677	425304.35

OPO: Output phase optimized. MEM: Espresso-MV memory over.

Table 2: Experimental result for adders with two-bit decoders.

			Proposed Partition			Random Partition 20 Iterations		Random Partition 50 Iterations	
Data	SOP	OPO SOP	EX-SOP	Time	OPO EX-SOP	EX-SOP	Time	EX-SOP	Time
adr4	17	14	13	0.09	12	13	1.64	13	4.10
adr5	26	22	18	0.34	18	18	5.81	18	15.64
adr6	37	32	25	0.81	25	25	18.39	25	51.90
adr7	50	44	33	1.94	33	33	58.91	33	136.55
adr8	65	58	42	6.62	42	43	181.41	43	429.76
adr9	82	74	52	26.11	52	51	655.76	51	1523.54
adr10	101	92	63	74.46	63	65	2433.61	61	5911.23
adr11	122	112	75	353.35	75	75	7918.99	75	23507.19

OPO: Output phase optimized.

experiments by using adders with two- and four-valued inputs and other benchmark functions with four-valued inputs. We obtained functions with four-valued inputs from their two-valued counterparts by pairing two variables using Espresso-MV. For all the experiments, we prepared minimized SOPs and output phase optimized SOPs by using Espresso-MV with default options.[†]

Tables 1 and 2 summarize the experimental data for adders with one- and two-bit decoders, respectively. To minimize EX-SOPs for adders we used: a) output phase optimized SOPs as the input for the EX-SOP minimizer; b) two different techniques to partition the cluster of cubes: partitioning method for adders from Section 5 and random partitioning method from AOXMIN [6]; and c) SIMPLIFY_LOCAL for SIMPLIFY_SINGLE and Espresso-MV for SIMPLIFY_DOUBLE in Fig. 3.

Table 1 shows that, for an n-bit adder with sufficiently large n, the proposed partitioning method is about 250 times faster to produce about two times better solution than the random partitioning method. Note that an SOP and an output phase optimized SOP for n-bit adder with two-valued inputs require $6 \cdot 2^n - 4n - 5$ and $5 \cdot 2^n - 4n - 3$ products, respectively [11]. However, from Table 1, we have the following:

[†]For all the tables in this section, the columns with heading 'SOP', 'OPO SOP', 'EX-SOP', and 'OPO EX-SOP' indicate the number of products in the corresponding expression, where 'OPO' is an abbreviation for 'output phase optimized'. Also, the columns with heading 'Time' indicate the CPU seconds spent by the extended version of AOXMIN to simplify EX-SOP and it do not include the time to prepare minimized SOP or output phase optimized SOP.

Conjecture 6.1 *When n is sufficiently large, an output phase optimized EX-SOP for n-bit adder with two-valued inputs requires at most 2^n products.*

The above shows that an output phase optimized EX-SOP requires about one sixth of the products in an SOP. This result would be useful to design adders.

Table 2 shows that the proposed partitioning method produced good solutions quickly. However, in most cases, these solutions can be obtained by random partitioning method by a reasonable increase in the computation time. The experimental data also reveals that the minimization time for EX-SOPs with four-valued inputs is much smaller than that for the corresponding EX-SOPs with two-valued inputs, because the former requires many fewer products than the later. Note that an EX-SOP for an n-bit adder with two-bit decoders requires at most $(n^2 + n + 2)/2$ products [13].

Table 3 presents experimental results for benchmark circuits with four-valued inputs. We used both SOPs and output phase optimized SOPs as the input for the EX-SOP minimizer; and Espresso-MV for both SIMPLIFY_SINGLE and SIMPLIFY_DOUBLE in Fig. 3.

We used adr6 to see how the choice of the two-level minimizers in Fig. 3 affect the quality of the solution and execution time. By using random partitions and 1000 iterations, we found that when Espresso-MV is used for both SIMPLIFY_SINGLE and SIMPLIFY_DOUBLE the algorithm requires 6253.79 seconds and produces a solution with 122 products; however, when we use SIMPLIFY_LOCAL for

Table 3: Experimental result for EX-SOPs with four-valued inputs.

	Input is SOP					Input is OPO SOP				
		20 Iterations		50 Iterations			20 Iterations		50 Iterations	
Data	SOP	EX-SOP	Time	EX-SOP	Time	OPO SOP	EX-SOP	Time	EX-SOP	Time
5xp1	46	35	19.31	35	51.15	42	29	14.89	29	39.56
addm4	109	97	208.03	95	519.51	101	89	168.09	89	404.82
f51m	51	37	17.86	35	47.13	50	40	18.16	37	46.20
life	26	20	4.72	20	11.86	26	20	4.74	20	12.43
rd84	54	35	22.81	35	57.72	37	31	37.30	31	92.09
rdm8	51	37	17.85	35	44.19	50	40	18.17	37	60.20
sqr8	157	152	274.39	147	674.02	148	139	212.63	139	577.24

OPO: Output phase optimized.

SIMPLIFY_SINGLE and Espresso-MV for SIMPLIFY_DOUBLE, the algorithm produces a solution with 81 products and requires 5956.69 seconds. We found similar tendencies for other adders too. However, it is our experience that, for many other benchmark functions, Espresso-MV for both SIMPLIFY_SINGLE and SIMPLIFY_DOUBLE is often a good choice.

7 Conclusions and Comments

EX-SOPs are promising because they often require many fewer products than SOPs. We demonstrated that, when n is sufficiently large, an n-bit adder with two-valued inputs requires at most 2^n products in an output phase optimized EX-SOP, while an output phase optimized SOP requires $5 \cdot 2^n - 4n - 3$ products. We presented partitioning method, which is very effective to optimize EX-SOPs for adders. Our experimental result shows that random partitioning method is unsuitable to design adders when n is large, because it requires excessive amount of CPU time to obtain a moderate design. For adders with two-bit decoders, proposed partitioning method is faster than random partitioning method in producing comparable solutions. We found that the choice of the two-level minimizers in AOXMIN-like-algorithm have a great influence on the number of products in EX-SOPs and a powerful minimizer is not always a good choice.

We obtained functions with four-valued inputs from their two-valued counterparts by pairing two variables using Espresso-MV code [10], which reduces the number of products in SOPs [11]. A different pairing algorithm targeting EX-SOPs may lead to better solutions. Currently, we are studying minimization of EX-SOPs for adders with carry inputs.

Acknowledgement

This work was supported in part by a Postdoctoral Fellowship of the Japan Society for the Promotion of Science and in part by a Grant-in-Aid for the Scientific Research of the Ministry of Education, Science, Culture, and Sports of Japan. We thank Prof. D. M. Miller and Dr. E. V. Dubrova for their valuable discussions on AOXMIN.

References

[1] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, 1984.

[2] D. Debnath and T. Sasao, "An optimization of AND-OR-EXOR three-level networks," *Proc. Asia and South Pacific Design Automation Conference*, pp. 545–550, Jan. 1997.

[3] D. Debnath and T. Sasao, "Exclusive-OR of two sum-of-products expressions: Simplification and an upper bound on the number of products," *Proc. 3rd International Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, Oxford, U.K., pp. 45–60, Sept. 1997.

[4] D. Debnath and T. Sasao, "A heuristic algorithm to design AND-OR-EXOR three-level networks," *Proc. Asia and South Pacific Design Automation Conference*, pp. 69–74, Feb. 1998.

[5] E. V. Dubrova, D. M. Miller, and J. C. Muzio, "Upper bounds on the number of products in AND-OR-XOR expansion of logic functions," *Electronics Letters*, Vol. 31, No. 7, pp. 541–542, Mar. 1995.

[6] E. V. Dubrova, D. M. Miller, and J. C. Muzio, "AOXMIN: A three-level heuristic AND-OR-XOR minimizer for Boolean functions," *Proc. 3rd International Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, Oxford, U.K., pp. 209–218, Sept. 1997.

[7] E. V. Dubrova and D. M. Miller, "Some experimental result of modified AOXMIN," Personal communication, May 1998.

[8] H. Fleisher, J. Giraldi, D. B. Martin, R. L. Phoenix, and M. A. Tavel, "Simulated annealing as a tool for logic optimization in a CAD environment," *Proc. International Conference on Computer-Aided Design*, pp. 203–205, Nov. 1985.

[9] D. Pellerin and M. Holley, *Practical Design Using Programmable Logic*, Prentice Hall, 1991.

[10] R. L. Rudell and A. Sangiovanni-Vincentelli, "Multiple-valued minimization for PLA optimization," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, Vol. CAD-6, No. 5, pp. 727–750, Sept. 1987.

[11] T. Sasao, "Input variable assignment and output phase optimization of PLA's," *IEEE Trans. Comput.*, Vol. C-33, No. 10, pp. 879–894, Oct. 1984.

[12] T. Sasao, "Logic synthesis with EXOR gates," in T. Sasao, ed., *Logic Synthesis and Optimization*, Kluwer Academic Publishers, 1993.

[13] T. Sasao, "A design method for AND-OR-EXOR three-level networks," *Proc. International Workshop on Logic Synthesis*, Lake Tahoe, California, pp. 8:11–8:20, May 1995.

[14] K. Shu, H. Yasuura, and S. Yajima, "Optimization of PLDs with output parity gates," *National Convention, Information Processing Society of Japan*, Mar. 1985 (in Japanese).

[15] A. Weinberger, "High-speed programmable logic array adders," *IBM Journal of Research and Development*, Vol. 23, No. 2, pp. 163–178, Mar. 1979.

The Output Permutation for the Multiple-Valued Logic Minimization with Universal Literals

Takahiro HOZUMI, Osamu KAKUSHO
Department of Economics & Information Science,
Hyogo University
2301, Shinzaike Hiraoka Kakogawa, 675-0101, JAPAN
hozumi@humans-kc.hyogo-dai.ac.jp

Yutaka HATA
Department of Computer Engineering,
Himeji Institute of Technology
2167, Shosha Himeji, 671-2201, JAPAN
hata@comp.eng.himeji-tech.ac.jp

Abstract

This paper shows the effectiveness of an output permutation for the implementation of current-mode CMOS circuits. A combination of a simple function and an output permutation can realize a difficult function and cost for the combination will be lower than the cost for a difficult function. The output permutation can be realized by a universal literal and we can calculate the cost. We first examine the all combinations of universal literals and output permutations and show that some combinations can realize one-variable functions with lower costs. Next, we minimize two-variable functions and compare the costs with the costs obtained by some output permutations. As the result, we show that about 70% functions can reduce the costs and their reduction ratio is about 12% on average.
Key words: logic synthesis, cost reduction, universal literal, output permutation, current-mode CMOS circuits.

1. Introduction

Recently, the use of current-mode CMOS circuits for implementation of multiple-valued logic functions has been studied by many researchers [1-5]. They have used universal literals to express logic functions and design their circuits based on cost tables. A cost table is a list of all literals with the corresponding implementation cost. Butler *et al.* showed two basic current-mode CMOS circuits and calculated the costs of some functions [1]. Based on the cost table, Lei *et al.* calculated the costs of all four-valued one-variable functions and proposed a minimization method for two-variable functions in 1991 [2]. Dueck *et al.* proposed a direct cover method in 1992 [4] and a minimization method using the simulated annealing in 1998 [5] based on the cost table of Ref. [2]. Moreover, in 1992, Lei *et al.* proposed the sharing of basic components and showed that costs for some one-variable functions can be reduced [3].

On the other hand, in the sum-of-products expression using window literals, Hata *et al.* showed the effectiveness of output permutation [6]. This paper showed that an output permutation could save about 10% product terms on average comparing with original expression. In the case of logic expression using universal literals, the cost depends on not only the number of product terms but also the universal literals forming the expression. We conclude that the output permutation can save the cost not only when the number of product terms is reduced by the output permutation but also when the number is not reduced. We therefore investigate the effectiveness of the output permutation for the logic expression using the universal literals.

In this paper, we investigate the effectiveness of the output permutations for the implementation of current-mode CMOS circuits. We first show that the cost of output permutation can be calculated by cost table of universal literals and show examples of cost reduction using it. Next, we examine all combinations of universal literals and output permutations, and show that some combinations can realize functions with lower costs. Finally, we minimize the two-variable functions and compare the costs with costs obtained by some output permutations. As the result, we show that about 70% functions can reduce the costs and its reduction ratio is about 12% on average.

2. Preliminaries

Let $P = \{0, 1, 2, ..., p-1\}$ be a set of p-valued truth values. Let $X=\{x_1, x_2, ..., x_n\}$ be a set of n variables, where x_i ($i = 1, 2, ..., n$) takes on values from P. An p-valued n-variable function $F(X)$ is a mapping $F:P^n \rightarrow P^1$. Each element in the domain of the function is said to be a *minterm* of the function.

A *universal literal* is a one-variable function and we denote it by $<a_0a_1...a_{p-1}>(x_i)$ instead of original notation $<a_0a_1...a_{p-1}>_{x_i}$. The universal literal is defined by

$$<a_0a_1...a_{p-1}>(x_i) = a_{x_i}.$$

A *product term*

$$A = <a_{10}a_{11}...a_{1\,p-1}>(x_1)\ <a_{20}a_{21}...a_{2\,p-1}>(x_2)\ ...<a_{n0}a_{n1}...a_{n\,p-1}>(x_n)$$

is defined as the minimum of its literals.

The *truncated sum* (TSUM for short) of two product terms A and B, expressed as $F = A + B$, is defined as

$$F(a) = \min(A(a)+B(a), p-1),$$

for all minterms. Any p-valued n-variable function can be expressed as truncated sum of product terms.

0-7695-0161-3/99 $10.00 © 1999 IEEE

Table 1 A cost table for all four-valued universal literals

	00	01	02	03	10	11	12	13	20	21	22	23	30	31	32	33
00xx	0	5	6	7	10	9	4	7	11	12	10	5	12	13	14	6
01xx	8	13	14	15	10	7	9	12	10	9	8	3	14	13	12	5
02xx	9	14	15	16	10	15	10	13	11	13	8	11	11	12	10	4
03xx	10	15	16	17	13	13	11	14	11	12	13	12	12	13	12	5
10xx	6	11	12	13	15	11	10	13	17	17	14	11	18	19	20	12
11xx	7	12	13	14	8	1	6	7	14	11	10	5	15	12	13	6
12xx	10	15	16	17	9	9	11	14	14	11	8	10	12	11	10	4
13xx	13	18	19	20	12	10	14	16	14	11	12	11	15	12	13	5
20xx	7	12	13	14	14	16	11	14	18	18	13	12	19	20	21	13
21xx	7	12	13	14	13	7	11	13	17	16	12	8	19	18	19	12
22xx	8	13	14	15	8	8	9	12	9	9	2	7	15	14	12	6
23xx	11	16	17	18	11	11	15	15	10	10	10	11	13	13	11	5
30xx	8	13	14	15	17	17	12	15	19	16	18	13	20	21	20	14
31xx	9	14	15	16	15	8	13	14	18	15	16	9	21	19	19	13
32xx	8	13	14	15	8	8	12	14	15	14	8	10	19	18	17	9
33xx	9	14	15	16	10	9	10	13	9	9	9	9	10	10	10	3

3. The output permutation and its effectiveness

A circuit for a function is implemented by current-mode CMOS circuit. We design the circuit by the logic expression using universal literals and Fig. 1 shows its circuit structure. The universal literals are composed of current source, transistor and so on, and their costs are calculated based on the number of transistors. Lei *et al.* tabulated the costs for the all four-valued universal literals in [2], and the table was updated in [3]. Table 1 shows the cost table for all four-valued universal literals [2-3]. In this table, first column indicates the output values for input values 0 and 1, and first row indicates the output values for 2 and 3. Using this table, we can calculate the circuit costs, where cost for MIN function is calculated as 5 according to [2].

As shown in the table, the each of the universal literals has a certain cost and there is a tendency that the cost of literal is low if its function is monotone between the costs of universal literals and their functions. The total cost of circuit composed of such low-cost literals is small. However, literals for a function is almost determined by the function. Then, we propose an output permutation as a method to transform the original function and to use lower-cost literals. The circuits can be realized by a structure as shown in the Fig. 2. The structure will be effective for reduction of circuit costs, although it can't use in time-critical applications because of increase of the circuit levels.

The output permutation has some cost and the combination of the transformed function and the permutation might be more costly than the direct implementation of original function. Then, we must

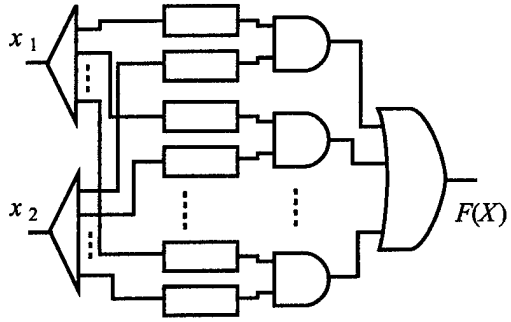


Fig. 1 A conventional circuit structure.

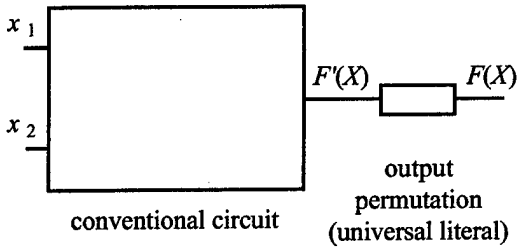


Fig. 2 A proposed circuit structure

evaluate their total costs to show the effectiveness of the output permutations. The output permutation can be realized by a universal literal, because it is a one-variable function. Then, we can calculate their cost using the cost table of universal literals. We show some examples.

Example 1: The logic function as shown in Fig. 3 can be expressed by

$$F(X) = <0012>(x_1) + <3010>(x_1)\ <0312>(x_2) + <0300>(x_1)\ <3122>(x_2)$$

and its cost is 68. The combination of a function as shown in Fig. 4 and output permutation <0312>(F') can realize the function of Fig. 3. Then, the function of Fig. 3 also can be expressed by

$$F'(X) = <0123>(x_1) + <0123>(x_2)$$
$$F(F') = <0312>(F')$$

and their total cost is 17 (6+11). The function of Fig. 4 can be derived by using the inverse function of <0312>(F'). Figure 5 shows the permutation and its inverse. (End of example)

Example 2: The logic function as shown in Fig. 6 can be expressed by

$$F(X) = <3100>(x_1)\ <1133>(x_2) + <0130>(x_1)\ <0310>(x_2) + <0012>(x_1)\ <0021>(x_2)$$

and its cost is 73 (=9+6+14+13+4+12+5*3). The combination of a function as shown in Fig. 7 and output permutation <2301>(F') can realize the function of Fig. 6. Then, the function of Fig. 6 also can be expressed by

$$F'(X) = <0133>(x_1)\ <3000>(x_2) + <2103>(x_1)\ <3300>(x_2) + <0233>(x_1)\ <0012>(x_2)$$
$$F(F') = <3210>(F')$$

and their total cost is 67 (=5+8+14+9+4+4+5*3+8). (End of example)

Example 1 is an extreme case. In this case, the number of product terms is reduced by an output permutation and it causes the cost reduction. In the case of Example 2, the number of product terms is not reduced, but the universal literals requiring lower costs are used with an output permutation. Thus, the cost can be reduced by the output permutation.

First, we investigated the effectiveness of the output permutations for one-variable functions. We have combined a universal literal and an output permutation, and have derived a function obtained by them. We have compared the total cost of the universal literal and the output permutation with the cost for direct implementation of the obtained function. We have done them for all combinations of universal literals and output permutations, and have listed all combinations which requiring lower costs than direct implementation. Table 2 shows the result. The saved costs for most of the functions have been 1 as shown in this table, but we consider that it is remarkable that the output permutations can reduce the costs on even one-variable functions.

x_2 \ x_1	0	1	2	3
0	0	3	1	2
1	3	1	2	2
2	1	2	2	2
3	2	2	2	2

Fig. 3 A function of example 1.

x_2 \ x_1	0	1	2	3
0	0	1	2	3
1	1	2	3	3
2	2	3	3	3
3	3	3	3	3

Fig. 4 A transformed function of Fig. 3.

F'	F
0	0
1	3
2	1
3	2

(a) <0312>(F')

F	F'
0	0
1	2
2	3
3	1

(b) Its inverse

Fig. 5 The output permutation and its inverse.

x_2 \ x_1	0	1	2	3
0	1	1	0	0
1	1	2	3	0
2	3	2	2	2
3	3	1	1	1

Fig. 6 A function of example 2.

x_2 \ x_1	0	1	2	3
0	2	2	3	3
1	2	1	0	3
2	0	1	1	1
3	0	2	2	2

Fig. 7 A transformed function of Fig. 6.

Table 2 Combinations requiring lower costs than direct implementation.

combination	total cost	function	cost	saved cost
<0012>(<0133>)	9 (4+ 5)	<0022>	10	1
<0121>(<0233>)	13 (9+ 4)	<0211>	15	2
<0033>(<0212>)	16 (6+10)	<0303>	17	1
<3200>(<0212>)	18 (8+10)	<3020>	19	1
<3012>(<0133>)	17 (12+ 5)	<3022>	18	1
<0300>(<1210>)	19 (10+ 9)	<3030>	20	1
<3312>(<0233>)	14 (10+ 4)	<3122>	16	2
<0312>(<1210>)	20 (11+ 9)	<3130>	21	1
<0033>(<2212>)	15 (6+ 9)	<3303>	16	1

Table 3 Costs of direct implementation and those with output permutation by our CAD tools.

	number	average cost		saving ratio
		original function	with permutation	
all functions	1000	89.7	81.8	8.7
cost-saved functions	686	94.3	82.8	12.2

Table 4 The number of times which each permutation is used by our CAD tools.

permutation	cost	times	permutation	cost	times	permutation	cost	times
none	—	314	<2103>	14	34	<0213>	13	21
<3210>	8	59	<0312>	11	32	<2301>	16	19
<2310>	11	57	<2013>	14	32	<3102>	15	18
<1023>	11	56	<1320>	14	31	<3120>	18	12
<0321>	12	54	<3201>	13	31	<2130>	19	9
<3012>	12	49	<1203>	17	25	<3021>	16	9
<0132>	12	40	<0231>	12	24	<1032>	20	7
<1230>	12	38	<1302>	19	23	<2031>	20	6

Table 5 Costs of direct implementation and those with output permutation by the CAD tools of Ref. [2].

	number	average cost		saving ratio
		original function	with permutation	
all functions	1000	95.2	86.3	9.3
cost-saved functions	722	99.6	87.4	12.3

Table 6 The number of times which each permutation is used by the CAD tools of Ref. [2].

permutation	cost	times	permutation	cost	times	permutation	cost	times
none	—	278	<3012>	12	41	<1203>	17	24
<3210>	8	55	<1320>	14	39	<3102>	15	24
<0312>	11	53	<3201>	13	37	<1302>	19	17
<2310>	11	48	<0231>	12	35	<2031>	20	15
<1023>	11	48	<0213>	13	28	<2130>	19	15
<2013>	14	47	<0132>	12	27	<3021>	16	14
<1230>	12	46	<2301>	16	27	<3120>	18	8
<0321>	12	44	<2103>	14	26	<1032>	20	4

The all functions of Table 2 consist of two or three kinds of logic values and it allows the combination to be low cost. However, such universal literals can't output all kinds of logic values. Most of two-or-more-variable logic functions have all logic values on their truth tables and functions not having them are few. Therefore, we use universal literals having all logic values as an output permutation except for <0123> and investigate the effectiveness of them for two-variable functions.

The best output permutation for a function can be obtained as follows. First, we minimize a given function and calculate the cost. Second, for each of all output permutations, the number of kinds is 23 (= 4! - 1), we derive the inverse permutation and transform the given function using it. We minimize the transformed function and calculate the total cost of the function and its output permutation. Finally, we compare the costs and select the combination requiring lowest costs.

According to the above, we have examined the randomly generated 1000 four-valued two-variable functions having all logic values, and show the result in Table 3. The table shows that we can reduce the cost for 686 functions out of 1000 functions and their saved cost is about 11.5 (12.2%) on average. Moreover, we show the number of times which each output permutation is used in Table 4. This table shows that the cost of main seven permutations are less than or equal to 12, and shows that the greater cost of universal literal is, the fewer times the literal is used. The universal literals whose cost is less than or equal to 12 are <3210>, <2310>, <1023>, <0321>, <3012>, <0132>, <1230>, <0312>, <0231>, and total times which the nine literals are used is 409. In addition, the 314 functions do not need an output permutation. Then, we can expect to obtain the best solutions for about 70% functions by minimizing a function ten times, no permutation and above nine permutations. The minimization of the above is done by GA based CAD tool made by us, while the similar results are obtained when we minimize them by Ref. [2] based CAD tool and Tables 5 and 6 show the result. From the tables, we can say that our CAD tools derive lower solution than the CAD tools of Ref. [2] and the cost reduction is not due to low minimization efficiency of the CAD tools. Then, output permutation is effective for the cost reduction of current-mode CMOS circuit implementations.

4. Conclusions

In this paper, we show the effectiveness of output permutation for the implementation of current-mode CMOS circuits. We investigated the combinations which require the lower costs than the direct implementation of a given function. As the result, we have shown that we can reduce their cost for 686 functions out of randomly generated 1000 four-valued two-variable functions and the ratio of their saved cost is 12.2% on average.

It remains as future studies to clear the effectiveness for three-or-more variable functions and investigate a method to find the best output permutation directly.

References

[1] S. Onneweer, H. Kerkhoff and J. T. Butler, "Structural computer-aided design of current-mode CMOS logic circuits," in *Proc. of 18th Int. Symp. on Multiple-Valued Logic*, pp.21-30, 1988.

[2] K. Lei and Z. G. Vranesic, "On the synthesis of 4-valued current mode CMOS circuits," in *Proc. of 21st Int. Symp. on Multiple-Valued Logic*, pp.147-155, 1991.

[3] K. Lei and Z. G. Vranesic, "Towards the realization of 4-valued CMOS circuits," in *Proc. of 22nd Int. Symp. on Multiple-Valued Logic*, pp.104-110, 1992.

[4] G. W. Dueck, "Direct cover MVL minimization with cost-tables," in *Proc. of 22nd Int. Symp. on Multiple-Valued Logic*, pp.58-65, 1992.

[5] B. Fraser and G. W. Dueck, "Multiple-valued logic minimization using universal literals and cost table," in *Proc. of 28th Int. Symp. on Multiple-Valued Logic*, pp.239-244, 1998.

[6] Y. Hata and K. Yamato, "Output permutation and the maximum number of implicants needed to cover the multiple-valued logic functions," *IEICE Trans. on Inf. and Syst.*, Vol. E76-D, No. 5, pp.555-561, 1993

Logical Model for Representing Uncertain Statuses of Multiple-Valued Logic Systems Realized by Min, Max and Literals

Noboru Takagi Akimitsu Hon-nami Kyoichi Nakashima

Department of Electronics and Informatics
Toyama Prefectural University
Kosugi, Toyama 939-0398, JAPAN
takagi@pu-toyama.ac.jp

Abstract

This paper will discuss on a logical model which is suitable for representing uncertain statuses existing in multiple-valued logic systems realized by minimum, maximum and literals. Then, some of the mathematical properties of functions defined by the logical model will be presented.

1 Introduction

S. C. Kleene [1] first defined regularity in the propositional operations of ternary logic $\{0, 1, \mathrm{u}\}$ as follows. A truth table for a ternary operation is *regular* if it satisfies the condition "*A given column (row) contains* 1 *in the* u *row (column), only if the column (row) consists entirely of* 1*'s; and likewise for* 0". Table 1 shows truth tables of regular ternary operations given by binary AND, OR and NOT operations, respectively. In Kleene's regularity, 0 (false) and 1 (true) are identical with the truth values of binary logic; while u is the third truth value whether it is true (1) or false (0) is uncertain or undefined. Therefore, the set-values $\{0\}$ and $\{1\}$ can be assigned to 0 and 1, respectively. Moreover, since the set-value $\{0, 1\}$ consist of the elements of the specific truth values that u may be when it is defined, the set-value $\{0, 1\}$ can be assigned to u. This suggests that basic operations of r-valued logic can be expanded into r-valued set-valued logic in a similar way as Kleene did when $r = 2$.

M. Mukaidono have proven that a ternary function $f : \{0, 1, \mathrm{u}\}^n \to \{0, 1, \mathrm{u}\}$ can be represented by means of the regular operations defined by Table 1 and the constants 0, 1 and u if and only if f is monotonic in the partial order relation $\prec$, which is defined by $0 \prec \mathrm{u}$, $1 \prec \mathrm{u}$ and $a \prec a$ where $a \in \{0, 1, \mathrm{u}\}$ [2]. Figure 1 shows a Hasse diagram of the partial order relation $\prec$. It is easy to verify that the partial order set $\langle \{0, 1, \mathrm{u}\}, \prec \rangle$ is isomorphic to the partial order set $\langle \{\{0\}, \{1\}, \{0, 1\}\}, \subseteq \rangle$ (See Figures 1 and 2), where $\subseteq$ is the set-theoretical inclusion. M. Mukaidono suggested an idea how functions on $\{0, 1, \mathrm{u}\}$ monotonic in $\prec$ can be expanded into $(2^r - 1)$-valued functions (where $r \geq 3$) [3]. Then, a way for expanding operations on $\{0, \ldots, r-1\}$ into those on the set of all nonempty subsets of $\{0, \ldots, r-1\}$ (V_r say) will be presented [6]. Operations on V_r given by this way will be called regular because it is an expansion of Kleene's regularity into $r \geq 3$. Now, minimum, maximum and literals play an important role in multiple-valued logic because they are functionally complete with constants on the set $\{0, \ldots, r-1\}$ [4]. This paper will first introduce regular operations on V_r given by minimum, maximum and literals on $\{0, \ldots, r-1\}$. Then, some of the mathematical properties of functions on V_r consisting of the regular operations will be presented.

Kleene's ternary logic is useful for representing and analyzing uncertain statuses of binary logic systems such as transient behavior of binary logic circuits [5]. Therefore, our results are useful for analyzing uncertain statuses of multiple-valued logic systems such as transient behavior of multiple-valued logic circuits realized by minimum, maximum and literals.

Table 1: Truth Tables of Regular Operations NOT, AND and OR

NOT		AND			OR		
		0	1	u	0	1	u
0	1	0	0	0	0	1	u
1	0	0	1	u	1	1	1
u	u	0	u	u	u	1	u

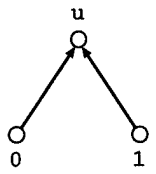


Fig. 1. Partial Order Set $\langle \{0, 1, u\}, \prec \rangle$

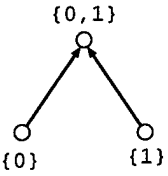


Fig. 2. Partial Order Set $\langle \{\{0\}, \{1\}, \{0, 1\}\}, \subseteq \rangle$

0-7695-0161-3/99 $10.00 © 1999 IEEE

2 Definitions and Properties of Functions Represented by Formulas

Let r be an r-valued set $\{0,\ldots,r-1\}$, and let V_r be the set of all nonempty subsets of r, i.e., $V_r \equiv \mathcal{P}(r)\backslash\{\emptyset\}$, where $\mathcal{P}(r)$ is the power set of r. If a subset of r consists of only one element, then it will be called a singleton. The set of all singletons of r will be denoted by r_s, i.e., $r_s \equiv \{\{0\},\ldots,\{r-1\}\}$. It is evident that the set V_r will be a partial order set when the set-theoretical inclusion $\subseteq$ employs in it. Figure 3 shows a Hasse diagram of the partial order set $\langle V_r, \subseteq\rangle$ when r is equal to 3.

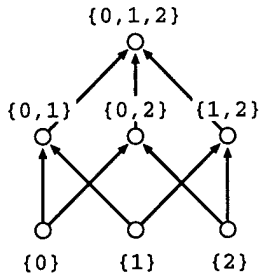


Fig. 3. Partial Order Set $\langle V_r, \subseteq\rangle$ where $r = 3$

Now, let us expand Kleene's regularity into operations on V_r below.

Definition 1: Let o be an n-ary operation on r. Then, an n-ary operation $\hat{o}$ on V_r based on o is defined by the following way: for any $a_1 \equiv \{a_1^1,\ldots,a_1^{m_1}\}$, ..., $a_n \equiv \{a_n^1,\ldots,a_n^{m_n}\}$ of V_r, $\hat{o}(a_1,\ldots,a_n) \equiv \bigcup_{i_1=1}^{m_1}\cdots\bigcup_{i_n=1}^{m_n}\{o(a_1^{i_1},\ldots,a_n^{i_n})\}$; and $\hat{o}$ will be called a *regular* operation on V_r.

The following three types of operations are basic in multiple-valued logic because r-valued functions consisting of these operations and the constants $0,\ldots,r-1$ are functionally complete on the set r [4].

$$a\cdot b \equiv \min(a,b),$$
$$a+b \equiv \max(a,b),$$
$$x^a \equiv \begin{cases} r-1 & \text{if } x=a,\\ 0 & \text{otherwise,}\end{cases}$$

where $a, b \in r$. The unary operations x^a are often called literals. Tables 2, 3 and 4 are truth table of regular operations on V_r based on $\cdot$, $+$ and x^a, respectively, when $r = 3$. Because the paper will focus on the regular operations of Tables 2, 3 and 4, they will be denoted by $\wedge$, $\vee$ and $x^{\{a\}}$, respectively.

Definition 2: Formulas are defined inductively as follows:

(1) Constants $\{0\},\ldots,\{r-1\}$ and literals x_i^a $(i = 1,\ldots,n; a \in r_s)$ are formulas.

Table 2: Truth Table of $\wedge$

$x\backslash y$	{0}	{1}	{2}	{0,1}	{0,2}	{1,2}	{0,1,2}
{0}	{0}	{0}	{0}	{0}	{0}	{0}	{0}
{1}	{0}	{1}	{1}	{0,1}	{0,1}	{1}	{0,1}
{2}	{0}	{1}	{2}	{0,1}	{0,2}	{1,2}	{0,1,2}
{0,1}	{0}	{0,1}	{0,1}	{0,1}	{0,1}	{0,1}	{0,1}
{0,2}	{0}	{0,1}	{0,2}	{0,1}	{0,2}	{0,1,2}	{0,1,2}
{1,2}	{0}	{1}	{1,2}	{0,1}	{0,1,2}	{1,2}	{0,1,2}
{0,1,2}	{0}	{0,1}	{0,1,2}	{0,1}	{0,1,2}	{0,1,2}	{0,1,2}

Table 3: Truth Table of $\vee$

$x\backslash y$	{0}	{1}	{2}	{0,1}	{0,2}	{1,2}	{0,1,2}
{0}	{0}	{1}	{2}	{0,1}	{0,2}	{1,2}	{0,1,2}
{1}	{1}	{1}	{2}	{1}	{1,2}	{1,2}	{1,2}
{2}	{2}	{2}	{2}	{2}	{2}	{2}	{2}
{0,1}	{0,1}	{1}	{2}	{0,1}	{0,1,2}	{1,2}	{0,1,2}
{0,2}	{0,2}	{1,2}	{2}	{0,1,2}	{0,2}	{1,2}	{0,1,2}
{1,2}	{1,2}	{1,2}	{2}	{1,2}	{1,2}	{1,2}	{1,2}
{0,1,2}	{0,1,2}	{1,2}	{2}	{0,1,2}	{0,1,2}	{1,2}	{0,1,2}

Table 4: Truth Table of x^a

x	{0}	{1}	{2}	{0,1}	{0,2}	{1,2}	{0,1,2}
$x^{\{0\}}$	{2}	{0}	{0}	{0,2}	{0,2}	{0}	{0,2}
$x^{\{1\}}$	{0}	{2}	{0}	{0,2}	{0}	{0,2}	{0,2}
$x^{\{2\}}$	{0}	{0}	{2}	{0}	{0,2}	{0,2}	{0,2}

(2) If g and h are formulas, then $(g \wedge h)$ and $(g \vee h)$ are also formulas.

(3) It is a formula if and only if we get it from (1) and (2) in a finite number of steps.

For the sake of simplicity, the operation $\wedge$ will be often omitted in formulas.

It will be considered that every formula can realize a function on V_r when each variable x_i will take a value of the set V_r. Furthermore, it is easy to verify that functions on V_r represented by formulas can not realize all of the functions on V_r, i.e., they are not functionally complete on V_r. Thus, one of the main subjects of the paper is to appear what functions on V_r can be realized by formulas.

Theorem 1: Let f be a function from V_r^n into V_r represented by a formula. Then, for any element $(a_1,\ldots,a_n) \in r_s^n$, $f(a_1,\ldots,a_n)$ is also an element of r_s.

Proof: The theorem can be proven directly from Definitions 1 and 2. ■

Theorem 2: Let f be a function from V_r^n into V_r represented by a formula. Then, $f(a_1,\ldots,a_n) \subseteq f(b_1,\ldots,b_n)$ holds for any elements $(a_1,\ldots,a_n)$ and $(b_1,\ldots,b_n)$ of V_r^n such that $a_i \subseteq b_i$ for all i.

Proof: It is evident that any constants $\{0\},\ldots,\{r-1\}$ and literals x_i^a (where $a \in r_s$) satisfy the theorem. Now, let us suppose that functions g and h on V_r represented by formulas satisfy the theorem, i.e., $g(a_1,\ldots,a_n) \subseteq g(b_1,\ldots,b_n)$ and $h(a_1,\ldots,a_n) \subseteq h(b_1,\ldots,b_n)$ for elements $(a_1,\ldots,a_n)$ and $(b_1,\ldots,b_n)$ of V_r^n such that $a_i \subseteq b_i$ for all i. Then, we can easily verify by Defi-

nition 1 that

$$g(a_1,\ldots,a_n)\wedge h(a_1,\ldots,a_n) \subseteq g(b_1,\ldots,b_n)\wedge h(b_1,\ldots,b_n) \text{ and}$$

$$g(a_1,\ldots,a_n)\vee h(a_1,\ldots,a_n) \subseteq g(b_1,\ldots,b_n)\vee h(b_1,\ldots,b_n).$$

This will complete the proof of the theorem because of the induction of the number of operations. ■

Theorem 3: Let f be a function from V_r^n into V_r represented by a formula, and let $a_1,\ldots,a_{i-1},a_{i+1},\ldots,a_n$ be any fixed elements of V_r. Then, for any a and b of $V_r^n\backslash r_s$, the least element of the subset $f(a_1,\ldots,a_{i-1},a,a_{i+1},\ldots,a_n)$ of r is equal to the least element of the subset $f(a_1,\ldots,a_{i-1},b,a_{i+1},\ldots,a_n)$, i.e.,

$$\min f(a_1,\ldots,a_{i-1},a,a_{i+1},\ldots,a_n) = \min f(a_1,\ldots,a_{i-1},b,a_{i+1},\ldots,a_n).$$

Proof: It is evident that any constants $\{0\},\ldots,\{r-1\}$ and literals x_i^a (where $a\in r_s$) satisfy the theorem. Now, let $s\equiv\{s_0,\ldots,s_m\}$ and $t\equiv\{t_0,\ldots,t_l\}$ be any elements of V_r such that $s_0<\cdots<s_m$ and $t_0<\cdots<t_l$. Then, by Definition 1, it follows that

$$s\wedge t=\bigcup_{i=0}^{m}\bigcup_{j=0}^{l}\{s_i\cdot t_j\} \text{ and } s\vee t=\bigcup_{i=0}^{m}\bigcup_{j=0}^{l}\{s_i+t_j\}.$$

Therefore, the least elements of $s\wedge t$ and $s\vee t$ are determined by the least elements of s and t. Thus, we will be able to prove the theorem by the induction of the number of operations. ■

From Theorems 1, 2 and 3, any function on V_r represented by a formula satisfies the following Condition A.

Condition A:

1. If $(a_1,\ldots,a_n)\in r_s^n$, then $f(a_1,\ldots,a_n)\in r_s$.
2. For any $(a_1,\ldots,a_n)$ and $(b_1,\ldots,b_n)$ of V_r^n, if $(a_1,\ldots,a_n)\subseteq(b_1,\ldots,b_n)$, then $f(a_1,\ldots,a_n)\subseteq f(b_1,\ldots,b_n)$.
3. For any a and b of $V_r^n\backslash r_s$, the least element of the subset $f(a_1,\ldots,a_{i-1},a,a_{i+1},\ldots,a_n)$ of r is equal to the least element of the subset $f(a_1,\ldots,a_{i-1},b,a_{i+1},\ldots,a_n)$, i.e.,

$$\min f(a_1,\ldots,a_{i-1},a,a_{i+1},\ldots,a_n) = \min f(a_1,\ldots,a_{i-1},b,a_{i+1},\ldots,a_n),$$

where $a_1,\ldots,a_{i-1},a_{i+1},\ldots,a_n$ are any fixed elements of V_r.

In the remainder of the paper, it will be proven that Condition A is a necessary and sufficient condition for a function on V_r to be represented by a formula.

Reference [3] has proven a necessary and sufficient condition for a function on $\{0,1/2,1\}$ to consist of the variables $x_1,\ldots,x_n$, operations min, max, $x\mapsto 1-x$ and the constants 0 and 1 (in which the constant 1/2 is excluded). The condition of Reference [3] is identical with the first and second conditions of Condition A when 0, 1 and 1/2 will refer to $\{0\}$, $\{1\}$ and $\{0,1\}$, respectively. That is, the third condition can be removed when $r=2$. This is because there exists only one element of $V_r\backslash r_s$ when $r=2$; and so the third condition A is trivial when $r=2$.

3 Properties between Functions Satisfying Condition A and Formulas

All of the proofs of Sections 3 and 4 will be removed because of the limitation of the paper.

The previous section have shown that any function on V_r represented by a formula satisfies Condition A. From now on, it will be proven that any function on V_r satisfying Condition A can be represented by a formula, and shown a way how a formula can be constructed from a given function on V_r satisfying Condition A. That is, Condition A will be a necessary and sufficient condition for a function on V_r to be represented by a formula.

This section will discuss some of the properties existing between functions satisfying Condition A and formulas.

Definition 3: Let f be a function from V_r^n into V_r, and let $A_i\equiv(a_1,\ldots,a_{i-1},a_{i+1},\ldots,a_n)$ be an element of r_s^{n-1}. Then, we will define two types of functions $f^{i+}_{a_1,\ldots,a_{i-1},a_{i+1},\ldots,a_n}(x)$ and $f^{i-}_{a_1,\ldots,a_{i-1},a_{i+1},\ldots,a_n}(x)$ $(i=1,\ldots,n)$ on V_r by the following formulas, respectively:

$$f^{i+}_{a_1,\ldots,a_{i-1},a_{i+1},\ldots,a_n}(x)\equiv\bigvee_{s\in r}\left\{\{s\}\wedge\bigvee_{b\in P^{i+}_{A_i}(s)}x^b\right\} \quad (1)$$

where $P^i_{A_i}(s)\equiv\{b\in V_r\,|\min f(a_1,\ldots,a_{i-1},b,a_{i+1}\cdots,a_n)=s\}$, and $P^{i+}_{A_i}(s)$ is the set of all maximal elements of the set $P^i_{A_i}(s)$; and

$$f^{i-}_{a_1,\ldots,a_{i-1},a_{i+1},\ldots,a_n}(x)\equiv$$

$$\bigvee_{s\in V_r\backslash r_s}\left(\bigvee_{t\in s}\left[\{t\}\wedge\left\{\bigvee_{b\in Q^{i-}_{A_i}(s)}\left(\bigwedge_{m\in b}x^{\{m\}}\right)\right\}\right]\right) \quad (2)$$

where $Q^i_{A_i}(s)\equiv\{b\in V_r\backslash r_s|f(a_1,\ldots,a_{i-1},b,a_{i+1}\cdots,a_n)=s\}$ and $Q^{i-}_{A_i}(s)$ is the set of all minimal elements of $Q^i_{A_i}(s)$.

In formulas (1) and (2), if $P^{i+}_{A_i}(s)$ and $Q^{i-}_{A_i}(s)$ are the emptyset, then

$$\bigvee_{b\in P^{i+}_{A_i}(s)}x^b \text{ and } \bigvee_{t\in s}\left[\{t\}\wedge\left\{\bigvee_{b\in Q^{i-}_{A_i}(s)}\left(\bigwedge_{m\in b}x^{\{m\}}\right)\right\}\right]$$

are defined by $\{0\}$; and if $P^{i+}_{A_i}(s)$ is $\{\{0,1,\ldots,r-1\}\}$, then $x^{\{0,1,\ldots,r-1\}}$ is defined by the constant r. Furthermore, if a function f on V_r satisfies Condition A and an element a of $V_r\backslash r_s$ belongs to a set $P^i_{A_i}(s)$, then the

element $\{0,1,\ldots,r-1\}$ of V_r has to also belong to the set $P^i_{A_i}(s)$ because f satisfy the second and the third conditions of Condition A. Thus, in this case, $P^{i+}_{A_i}(s)$ will be the set $\{\{0,1,\ldots,r-1\}\}$. Therefore, because $P^{i+}_{A_i}(s) = \{r\}$ or $P^{i+}_{A_i}(s) \subseteq \boldsymbol{r}_s$ holds for each $P^{i+}_{A_i}(s)$, formula (1) is well-defined when f satisfies Condition A.

Lemma 1: Let f be a function from V_r^n into V_r satisfying Condition A, and let $a_1,\ldots,a_{i-1},a_{i+1},\ldots,a_n$ be elements of $\boldsymbol{r}_s$. Then, for every $a \in V_r$, there exists a subset K of r such that $\{f_0\} \subseteq K \subseteq f(a_1,\ldots,a_{i-1},a,a_{i+1},\ldots,a_n)$ (in which f_0 is the least element of $f(a_1,\ldots,a_{i-1},a,a_{i+1},\ldots,a_n)$), and the following equality holds for the element a:

$$f^{i+}_{a_1,\ldots,a_{i-1},a_{i+1},\ldots a_n}(a) = \begin{cases} f(A_i(a)) & \text{if } f(A_i(a)) \in \boldsymbol{r}_s \\ K & \text{otherwise} \end{cases} \quad (3)$$

where $A_i(a) \equiv (a_1,\ldots,a_{i-1},a,a_{i+1},\ldots,a_n)$.

Lemma 2: Let f be a function from V_r^n into V_r satisfying Condition A, and let $a_1,\ldots,a_{i-1},a_{i+1},a_n$ be elements of $\boldsymbol{r}_s$. Then, the following equality holds for every $a \in V_r$:

$$f^{i-}_{a_1,\ldots,a_{i-1},a_{i+1},a_n}(a) = \begin{cases} \{0\} & \text{if } f(A_i(a)) \in \boldsymbol{r}_s \\ \{0\} \cup f(A_i(a)) & \text{otherwise} \end{cases} \quad (4)$$

where $A_i(a) \equiv (a_1,\ldots,a_{i-1},a,a_{i+1},\ldots,a_n)$.

Example 1: Consider the function f on V_3 defined by Table 5. It is easy to verify that f satisfies Condition A. Since

$$\begin{aligned} P^1_{\{0\}}(0) &= \{a \in V_3 \mid \min f(a,\{0\}) = 0\} \\ &= \{\{0\},\{2\},\{0,1\},\{0,2\},\{1,2\},\{0,1,2\}\}, \\ P^1_{\{0\}}(1) &= \{a \in V_3 \mid \min f(a,\{0\}) = 1\} = \{\{1\}\}, \\ P^1_{\{0\}}(2) &= \{a \in V_3 \mid \min f(a,\{0\}) = 2\} = \emptyset, \end{aligned}$$

we have $P^{1+}_{\{0\}}(0) = \{\{0,1,2\}\}$, $P^{1+}_{\{0\}}(1) = \{\{1\}\}$ and $P^{1+}_{\{0\}}(2) = \emptyset$. Therefore,

$$\begin{aligned} f^{1+}_{\{0\}}(x) &= (\{0\} \wedge \{0,1,2\}) \vee \left(\{1\} \wedge x^{\{1\}}\right) \vee (\{2\} \wedge \{0\}) \\ &= \{1\}x^{\{1\}}. \end{aligned} \quad (5)$$

In a similar way,

$$\left.\begin{aligned} f^{1+}_{\{1\}}(x) &= \{1\}x^{\{1\}} \vee x^{\{2\}}, \\ f^{1+}_{\{2\}}(x) &= x^{\{1\}}, \\ f^{2+}_{\{0\}}(y) &= \{0\}, \\ f^{2+}_{\{1\}}(y) &= \{1\} \vee y^{\{2\}}, \\ f^{2+}_{\{2\}}(y) &= y^{\{1\}}. \end{aligned}\right\} \quad (6)$$

Table 5: Example of Function f on V_3 Satisfying Condition A

$x\backslash y$	$\{0\}$	$\{1\}$	$\{2\}$	$\{0,1\}$	$\{0,2\}$	$\{1,2\}$	$\{0,1,2\}$
$\{0\}$	$\{0\}$	$\{0\}$	$\{0\}$	$\{0\}$	$\{0\}$	$\{0,2\}$	$\{0,2\}$
$\{1\}$	$\{1\}$	$\{1\}$	$\{2\}$	$\{1\}$	$\{1,2\}$	$\{1,2\}$	$\{1,2\}$
$\{2\}$	$\{0\}$	$\{2\}$	$\{0\}$	$\{0,2\}$	$\{0\}$	$\{0,2\}$	$\{0,2\}$
$\{0,1\}$	$\{0,1\}$	$\{0,1\}$	$\{0,2\}$	$\{0,1,2\}$	$\{0,1,2\}$	$\{0,1,2\}$	$\{0,1,2\}$
$\{0,2\}$	$\{0\}$	$\{0,2\}$	$\{0\}$	$\{0,2\}$	$\{0\}$	$\{0,2\}$	$\{0,2\}$
$\{1,2\}$	$\{0,1,2\}$	$\{0,1,2\}$	$\{0,2\}$	$\{0,1,2\}$	$\{0,1,2\}$	$\{0,1,2\}$	$\{0,1,2\}$
$\{0,1,2\}$	$\{0,1,2\}$	$\{0,1,2\}$	$\{0,2\}$	$\{0,1,2\}$	$\{0,1,2\}$	$\{0,1,2\}$	$\{0,1,2\}$

Moreover, since

$$\begin{aligned} Q^1_{\{0\}}(\{0,1\}) &= \{a \in V_3\backslash \mathbf{3}_s \mid f(a,\{0\}) = \{0,1\}\} \\ &= \{\{0,1\}\}, \\ Q^1_{\{0\}}(\{0,2\}) &= \{a \in V_3\backslash \mathbf{3}_s \mid f(a,\{0\}) = \{0,2\}\} = \emptyset, \\ Q^1_{\{0\}}(\{1,2\}) &= \{a \in V_3\backslash \mathbf{3}_s \mid f(a,\{0\}) = \{1,2\}\} = \emptyset, \\ Q^1_{\{0\}}(\{0,1,2\}) &= \{a \in V_3\backslash \mathbf{3}_s \mid f(a,\{0\}) = \{0,1,2\}\} \\ &= \{\{1,2\},\{0,1,2\}\}, \end{aligned}$$

we have $Q^{1-}_{\{0\}}(\{0,1\}) = \{\{0,1\}\}$, $Q^{1-}_{\{0\}}(\{0,2\}) = Q^{1-}_{\{0\}}(\{1,2\}) = \emptyset$ and $Q^{1-}_{\{0\}}(\{0,1,2\}) = \{\{1,2\}\}$. Therefore,

$$\begin{aligned} f^{1-}_{\{0\}}(x) &= \left\{\left(\{0\} \wedge x^{\{0\}}x^{\{1\}}\right) \vee \left(\{1\} \wedge x^{\{0\}}x^{\{1\}}\right)\right\} \vee \\ &\quad \{0\} \vee \{0\} \vee \left\{\left(\{0\} \wedge x^{\{2\}}x^{\{2\}}\right) \vee \right. \\ &\quad \left.\left(\{1\} \wedge x^{\{2\}}x^{\{2\}}\right) \vee \left(\{2\} \wedge x^{\{2\}}x^{\{2\}}\right)\right\} \\ &= \{1\}x^{\{0\}}x^{\{1\}} \vee \{1\}x^{\{1\}}x^{\{2\}} \vee x^{\{1\}}x^{\{2\}}. \end{aligned} \quad (7)$$

In a similar way,

$$\left.\begin{aligned} f^{1-}_{\{1\}}(x) &= \{1\}x^{\{0\}}x^{\{1\}} \vee x^{\{0\}}y^{\{2\}} \vee \{1\}x^{\{1\}}x^{\{2\}} \\ &\quad \vee x^{\{1\}}x^{\{2\}}, \\ f^{1-}_{\{2\}}(x) &= x^{\{0\}}x^{\{1\}} \vee x^{\{1\}}x^{\{2\}}, \\ f^{2-}_{\{0\}}(y) &= y^{\{1\}}y^{\{2\}}, \\ f^{2-}_{\{1\}}(y) &= \{1\}y^{\{0\}}y^{\{2\}} \vee x^{\{0\}}y^{\{2\}} \vee \{1\}y^{\{1\}}y^{\{2\}} \\ &\quad \vee y^{\{1\}}y^{\{2\}}, \\ f^{2-}_{\{2\}}(y) &= y^{\{0\}}y^{\{1\}} \vee y^{\{1\}}y^{\{2\}}. \end{aligned}\right\} \quad (8)$$

Table 6 shows truth tables of f^{i+}_a and f^{i-}_a for which $i = 1,2$ and $a \in \{\{0\},\{1\},\{2\}\}$.

Lemma 3: Let f be a function from V_r^n into V_r satisfying Condition A, and let $a_1,\ldots,a_{i-1},a_{i+1},\ldots,a_n$ be elements of $\boldsymbol{r}_s$. Then, the following equality holds for every $a \in V_r$:

$$\begin{aligned} &f^{i+}_{a_1,\ldots,a_{i-1},a_{i+1},\ldots,a_n}(a) \vee f^{i-}_{a_1,\ldots,a_{i-1},a_{i+1},\ldots,a_{n-1}}(a) \\ &\quad = f(A_i(a)), \end{aligned}$$

where $A_i(a) \equiv (a_1,\ldots,a_{i-1},a,a_{i+1},\ldots,a_n)$.

Table 6: Truth Table of f_a^{i+} and f_a^{i-} ($i = 1, 2$; $a \in \{\{0\}, \{1\}, \{2\}\}$)

x	$\{0\}$	$\{1\}$	$\{2\}$	$\{0,1\}$	$\{0,2\}$	$\{1,2\}$	$\{0,1,2\}$
$f_{\{0\}}^{1+}(x)$	$\{0\}$	$\{1\}$	$\{0\}$	$\{0,1\}$	$\{0\}$	$\{0,1\}$	$\{0,1\}$
$f_{\{1\}}^{1+}(x)$	$\{0\}$	$\{1\}$	$\{2\}$	$\{0,1\}$	$\{0,2\}$	$\{0,1,2\}$	$\{0,1,2\}$
$f_{\{2\}}^{1+}(x)$	$\{0\}$	$\{2\}$	$\{0\}$	$\{0,2\}$	$\{0\}$	$\{0,2\}$	$\{0,2\}$
$f_{\{0\}}^{1-}(x)$	$\{0\}$	$\{0\}$	$\{0\}$	$\{0,1\}$	$\{0\}$	$\{0,1,2\}$	$\{0,1,2\}$
$f_{\{1\}}^{1-}(x)$	$\{0\}$	$\{0\}$	$\{0\}$	$\{0,1\}$	$\{0,2\}$	$\{0,1,2\}$	$\{0,1,2\}$
$f_{\{2\}}^{1-}(x)$	$\{0\}$	$\{0\}$	$\{0\}$	$\{0,2\}$	$\{0\}$	$\{0,2\}$	$\{0,2\}$
$f_{\{0\}}^{2+}(x)$	$\{0\}$	$\{0\}$	$\{0\}$	$\{0\}$	$\{0\}$	$\{0\}$	$\{0\}$
$f_{\{1\}}^{2+}(x)$	$\{1\}$	$\{1\}$	$\{2\}$	$\{1\}$	$\{1,2\}$	$\{1,2\}$	$\{1,2\}$
$f_{\{2\}}^{2+}(x)$	$\{0\}$	$\{2\}$	$\{0\}$	$\{0,2\}$	$\{0\}$	$\{0,2\}$	$\{0,2\}$
$f_{\{0\}}^{2-}(x)$	$\{0\}$	$\{0\}$	$\{0\}$	$\{0\}$	$\{0\}$	$\{0,2\}$	$\{0,2\}$
$f_{\{1\}}^{2-}(x)$	$\{0\}$	$\{0\}$	$\{0\}$	$\{0\}$	$\{0,1,2\}$	$\{0,1,2\}$	$\{0,1,2\}$
$f_{\{2\}}^{2-}(x)$	$\{0\}$	$\{0\}$	$\{0\}$	$\{0,2\}$	$\{0\}$	$\{0,2\}$	$\{0,2\}$

Table 7: Truth Table of $f_a \equiv f_a^{i+} \vee f_a^{i-}$ ($i = 1, 2$; $a \in \{\{0\}, \{1\}, \{2\}\}$)

x	$\{0\}$	$\{1\}$	$\{2\}$	$\{0,1\}$	$\{0,2\}$	$\{1,2\}$	$\{0,1,2\}$
$f_{\{0\}}^{1}(x)$	$\{0\}$	$\{1\}$	$\{0\}$	$\{0,1\}$	$\{0\}$	$\{0,1,2\}$	$\{0,1,2\}$
$f_{\{1\}}^{1}(x)$	$\{0\}$	$\{1\}$	$\{2\}$	$\{0,1\}$	$\{0,2\}$	$\{0,1,2\}$	$\{0,1,2\}$
$f_{\{2\}}^{1}(x)$	$\{0\}$	$\{2\}$	$\{0\}$	$\{0,2\}$	$\{0\}$	$\{0,2\}$	$\{0,2\}$
$f_{\{0\}}^{2}(x)$	$\{0\}$	$\{0\}$	$\{0\}$	$\{0\}$	$\{0\}$	$\{0,2\}$	$\{0,2\}$
$f_{\{1\}}^{2}(x)$	$\{1\}$	$\{1\}$	$\{2\}$	$\{1\}$	$\{1,2\}$	$\{1,2\}$	$\{1,2\}$
$f_{\{2\}}^{2}(x)$	$\{0\}$	$\{2\}$	$\{0\}$	$\{0,2\}$	$\{0\}$	$\{0,2\}$	$\{0,2\}$

Example 2: Consider the function f defined by Table 5. By Lemma 3, it follows that

$$f(x,a)=f_a^{1+}(x) \vee f_a^{1-}(x), \quad \text{and}$$
$$f(a,y)=f_a^{2+}(y) \vee f_a^{2-}(y)$$

hold for every $a \in \{\{0\},\{1\},\{2\}\}$, where $f_a^{1+}(x)$, $f_a^{1-}(x)$, $f_a^{2+}(y)$ and $f_a^{2-}(y)$ have been given in equations (5), (6), (7) and (8). Table 7 shows truth tables of $f_a^{i+} \vee f_a^{i-}$ for which $i = 1, 2$ and $a \in \{\{0\},\{1\},\{2\}\}$.

4 Realization of Formulas for Functions Satisfying Condition A

This section will prove that any function on V_r satisfying Condition A can be represented by a formula, and also show a way how a formula can be constructed from a given function on V_r satisfying Condition A.

Definition 4: Let f be a function from V_r^n into V_r. Then, a function f_1 on V_r will be defined in the following formula:

$$f_1(x_1,\ldots,x_n) \equiv \bigvee_{i=1}^{n} f^i(x_1,\ldots,x_n),$$

where

$$f^i \equiv \bigvee_{(a_1,\ldots,a_{i-1},a_{i+1},\ldots,a_n)\in r_s^{n-1}} \left(\bigwedge_{j=1;j\neq i}^{n} x_j^{a_j} \wedge f^i_{a_1,\ldots,a_{i-1},a_{i+1},\ldots,a_n}(x_i) \right)$$

and $f^i_{a_1,\ldots,a_{i-1},a_{i+1},\ldots,a_n}(x) \equiv f^{i+}_{a_1,\ldots,a_{i-1},a_{i+1},\ldots,a_n}(x) \vee f^{i-}_{a_1,\ldots,a_{i-1},a_{i+1},\ldots,a_n}(x)$.

Here, a special subset $I(r,n)$ of V_r^n will be defined in the following way:

$$I(r,n) \equiv \bigcup_{i=1}^{n} \{(a_1,\ldots,a_n) \in V_r^n \mid a_i \in V_r \backslash r_s \text{ and } a_1,\ldots,a_{i-1},a_{i+1},\ldots,a_n \in r_s\}$$

That is, for every element $(a_1,\ldots,a_n)$ of $I(r,n)$, there exist only one element of $V_r \backslash r_s$ and $n-1$ elements of r_s.

Lemma 4: Let f be a function from V_r^n into V_r satisfying Condition A. Then, for every $(a_1,\ldots,a_n) \in V_r^n$, there exists a subset K of r such that $\{0\} \subseteq K \{0\} \cup f(a_1,\ldots,a_n)$, and the following equality holds for the element $(a_1,\ldots,a_n)$:

$$f_1(a_1,\ldots,a_n) = \begin{cases} f(a_1,\ldots,a_n) & \text{if } f(a_1,\ldots,a_n) \in r_s \cup I(n,r) \\ K & \text{otherwise.} \end{cases}$$

Example 3: Consider the function f defined by Table 5. Then, by equalities (5) and (6),

$$\begin{aligned} f_{\{0\}}^1(x) &= f_{\{0\}}^{1+}(x) \vee f_{\{0\}}^{1-}(x) \\ &= \{1\}x^{\{1\}} \vee \{1\}x^{\{0\}}x^{\{1\}} \vee \{1\}x^{\{1\}}x^{\{2\}} \vee x^{\{1\}}x^{\{2\}} \\ &= \{1\}x^{\{1\}} \vee x^{\{1\}}x^{\{2\}}. \end{aligned}$$

In a similar way, by equalities (6), (7) and (8),

$$\begin{aligned} f_{\{1\}}^1(x) &= \{1\}x^{\{1\}} \vee x^{\{2\}}, \\ f_{\{2\}}^1(x) &= x^{\{1\}}, \\ f_{\{0\}}^2(y) &= y^{\{1\}}y^{\{2\}}, \\ f_{\{1\}}^2(y) &= \{1\} \vee y^{\{2\}}, \\ f_{\{2\}}^2(y) &= y^{\{1\}}. \end{aligned}$$

Therefore, $f_1(x,y)$ of Definition 4 is given by the following formula:

$$f_1(x,y) = f^1(x,y) \vee f^2(x,y), \tag{9}$$

where

$$\begin{aligned} f^1 &= y^{\{0\}} f_{\{0\}}^1(x) \vee y^{\{1\}} f_{\{1\}}^1(x) \vee y^{\{2\}} f_{\{2\}}^1(x), \\ f^2 &= x^{\{0\}} f_{\{0\}}^2(y) \vee x^{\{1\}} f_{\{1\}}^2(y) \vee x^{\{2\}} f_{\{2\}}^2(y). \end{aligned}$$

Table 8: Truth Table of f_1

$x \backslash y$	$\{0\}$	$\{1\}$	$\{2\}$	$\{0,1\}$	$\{0,2\}$	$\{1,2\}$	$\{0,1,2\}$
$\{0\}$	$\{0\}$	$\{0\}$	$\{0\}$	$\{0\}$	$\{0\}$	$\{0,2\}$	$\{0,2\}$
$\{1\}$	$\{1\}$	$\{1\}$	$\{2\}$	$\{1\}$	$\{1,2\}$	$\{1,2\}$	$\{1,2\}$
$\{2\}$	$\{0\}$	$\{2\}$	$\{0\}$	$\{0,2\}$	$\{0\}$	$\{0,2\}$	$\{0,2\}$
$\{0,1\}$	$\{0,1\}$	$\{0,1\}$	$\{0,2\}$	$\{0,1\}$	$\{0,1,2\}$	$\{0,1,2\}$	$\{0,1,2\}$
$\{0,2\}$	$\{0\}$	$\{0,2\}$	$\{0\}$	$\{0,2\}$	$\{0\}$	$\{0,2\}$	$\{0,2\}$
$\{1,2\}$	$\{0,1,2\}$	$\{0,1,2\}$	$\{0,2\}$	$\{0,1,2\}$	$\{0,1,2\}$	$\{0,1,2\}$	$\{0,1,2\}$
$\{0,1,2\}$	$\{0,1,2\}$	$\{0,1,2\}$	$\{0,2\}$	$\{0,1,2\}$	$\{0,1,2\}$	$\{0,1,2\}$	$\{0,1,2\}$

Table 8 shows a truth table of $f_1(x,y)$.

Definition 5: Let f be a function from V_r^n into V_r. Then, a function f_2 on V_r will be defined in the following formula:

$$f_2(x_1,\ldots,x_n) \equiv$$

$$\{s_0\} \vee \left[\bigvee_{s \in V_r \setminus r_s} \left\{ \bigvee_{t \in s} \{t\} \wedge f_s(x_1,\ldots,x_n) \right\} \right],$$

where $f_s(x_1,\ldots,x_n) \equiv \bigvee_{(a_1,\ldots,a_n)\in T^-(s)} \left\{ \bigwedge_{b\in a_1} x_1^{\{b\}} \wedge \cdots \wedge \bigwedge_{b\in a_n} x_n^{\{b\}} \right\}$, $T(s)$ (where $s \in V_r \setminus r_s$) is the set of all elements of V_r^n such that $f(a_1,\ldots,a_n) = s$ and $(a_1,\ldots,a_n) \notin r_s^n \cup I$, $T^-(s)$ is the set of all minimal elements of $T(s)$, and s_0 is the least element of $\bigcup_{(a_1,\ldots,a_n)\in V_r^n} f(a_1,\ldots,a_n)$.

Lemma 5: Let f be a function from V_r^n into V_r satisfying Condition A. Then, for any element $(a_1,\ldots,a_n)$ of V_r^n,

$$f_2(a_1,\ldots,a_n) =$$

$$\begin{cases} \{s_0\} & \text{if } (a_1,\ldots,a_n) \in r_s^n \cup I(n,r) \\ f(a_1,\ldots,a_n) & \text{otherwise,} \end{cases}$$

where s_0 is the least element of the set $\bigcup_{(a_1,\ldots,a_n)\in V_r^n} f(a_1,\ldots,a_n)$.

Example 4: Consider the function f defined by Table 5. Since each $T^-(s)$ ($s \in \{\{0,1\},\{0,2\},\{1,2\},\{0,1,2\}\}$), is given by Table 9, each f_s of Definition 5 is expressed in the following formulas:

$$f_{\{0,1\}} = x^{\{0\}}x^{\{1\}}y^{\{0\}} \vee x^{\{0\}}x^{\{1\}}y^{\{1\}},$$

$$f_{\{0,2\}} = x^{\{0\}}y^{\{1\}}y^{\{2\}} \vee x^{\{2\}}y^{\{0\}}y^{\{1\}} \vee x^{\{2\}}y^{\{0\}}y^{\{2\}} \vee x^{\{0\}}y^{\{1\}}y^{\{2\}} \vee x^{\{0\}}x^{\{2\}}y^{\{1\}} \vee x^{\{1\}}x^{\{2\}}y^{\{2\}},$$

$$f_{\{1,2\}} = x^{\{1\}}y^{\{0\}}y^{\{2\}} \vee x^{\{1\}}y^{\{1\}}y^{\{2\}},$$

$$f_{\{0,1,2\}} = x^{\{1\}}x^{\{2\}}y^{\{0\}} \vee x^{\{1\}}x^{\{2\}}y^{\{1\}}.$$

Therefore, $f_2(x,y)$ of Definition 5 is given by

$$\begin{aligned} f_2(x,y) &= \{0\} \vee (\{0\}f_{\{0,1\}} \vee \{1\}f_{\{0,1\}}) \vee \\ &\quad (\{0\}f_{\{0,2\}} \vee \{2\}f_{\{0,2\}}) \vee (\{1\}f_{\{1,2\}} \vee \{2\}f_{\{1,2\}}) \vee \\ &\quad (\{0\}f_{\{0,1,2\}} \vee \{1\}f_{\{0,1,2\}} \vee \{2\}f_{\{0,1,2\}}) \\ &= \{1\}f_{\{0,1\}} \vee f_{\{0,2\}} \vee \{1\}f_{\{1,2\}} \vee f_{\{1,2\}} \vee \\ &\quad \{1\}f_{\{0,1,2\}} \vee f_{\{0,1,2\}} \end{aligned} \quad (10)$$

Table 10 shows a truth table of $f_2(x,y)$.

Table 9: $T(s)$ and $T^-(s)$ where $s \in \{\{0,1\}, \{0,2\}, \{1,2\}, \{0,1,2\}\}$: All of the elements (x,y) of V_3^2 corresponding to $T(s)$ are elements of $T(s)$; and all of the elements (x,y) of V_3^2 corresponding to $T^-(s)$ are elements of $T(s)$ and $T^-(s)$.

$x\backslash y$	$\{0,1\}$	$\{0,2\}$	$\{1,2\}$	$\{0,1,2\}$
$\{0\}$	–	–	$T^-(\{0,2\})$	$T(\{0,2\})$
$\{1\}$	–	$T^-(\{1,2\})$	$T^-(\{1,2\})$	$T(\{1,2\})$
$\{2\}$	$T^-(\{0,2\})$	–	$T^-(\{0,2\})$	$T(\{0,2\})$

$x\backslash y$	$\{0\}$	$\{1\}$	$\{2\}$
$\{0,1\}$	$T^-(\{0,1\})$	$T^-(\{0,1\})$	$T^-(\{0,2\})$
$\{0,2\}$	–	$T^-(\{0,2\})$	–
$\{1,2\}$	$T^-(\{0,1,2\})$	$T^-(\{0,1,2\})$	$T^-(\{0,2\})$
$\{0,1,2\}$	$T(\{0,1,2\})$	$T(\{0,1,2\})$	$T(\{0,2\})$

Table 10: Truth Table of $f_2(x,y)$

$x\backslash y$	$\{0\}$	$\{1\}$	$\{2\}$	$\{0,1\}$	$\{0,2\}$	$\{1,2\}$	$\{0,1,2\}$
$\{0\}$	$\{0\}$	$\{0\}$	$\{0\}$	$\{0\}$	$\{0\}$	$\{0\}$	$\{0\}$
$\{1\}$	$\{0\}$	$\{0\}$	$\{0\}$	$\{0\}$	$\{0\}$	$\{0\}$	$\{0\}$
$\{2\}$	$\{0\}$	$\{0\}$	$\{0\}$	$\{0\}$	$\{0\}$	$\{0\}$	$\{0\}$
$\{0,1\}$	$\{0\}$	$\{0\}$	$\{0\}$	$\{0,1,2\}$	$\{0,1,2\}$	$\{0,1,2\}$	$\{0,1,2\}$
$\{0,2\}$	$\{0\}$	$\{0\}$	$\{0\}$	$\{0,2\}$	$\{0\}$	$\{0,2\}$	$\{0,2\}$
$\{1,2\}$	$\{0\}$	$\{0\}$	$\{0\}$	$\{0,1,2\}$	$\{0,1,2\}$	$\{0,1,2\}$	$\{0,1,2\}$
$\{0,1,2\}$	$\{0\}$	$\{0\}$	$\{0\}$	$\{0,1,2\}$	$\{0,1,2\}$	$\{0,1,2\}$	$\{0,1,2\}$

Theorem 4: Let f be a function from V_r^n into V_r satisfying Condition A. Then, for every $(a_1,\ldots,a_n)$ of V_r^n,
$f(a_1,\ldots,a_n) = f_1(a_1,\ldots,a_n) \vee f_2(a_1,\ldots,a_n)$,
where f_1 and f_2 are formulas consisting of f by Definitions 4 and 5.

By Theorem 4, it follows that the function f of Table 5, which satisfies Condition A, is represented by the formulas $f_1(x,y)$ and $f_2(x,y)$ of (9) and (10).

5 Conclusions

This paper have discussed an extension of Kleene's regularity into V_r. Then, a necessary and sufficient condition for a function on V_r to be realized by $\wedge$, $\vee$, $x^{\{a\}}$ (where $a \in r$) and the constants $\{0\},\ldots,\{r-1\}$ has been proven.

References

[1] S. C. Kleene, *Introduction to metamathematics*, North-Holland Pub., pp. 332-340, 1952.

[2] M. Mukaidono, "Regular ternary logic functions – Ternary logic functions suitable for treating ambiguity," *IEEE Trans. on Computers*, vol. c-35, no. 2, pp. 179-183, 1986.

[3] M. Mukaidono, "On the B-ternary logical function," *IECE Trans.*, vol. 55-D, no. 6, pp. 355-362, 1972 (in Japanese)

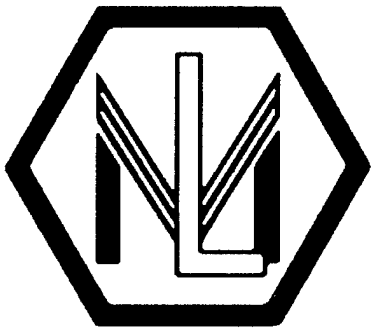
[4] J. C. Muzio and T. C. Wesselkamper, *Multiple-Valued Switching Theory*.

[5] M. Mukaidono: "The B-ternary logic and its applications to the detection of hazards in combinational switching circuits, " *Proc. of the 8th ISMVL*, pp. 269-275, 1978.

[6] N. Takagi, Y. Nakamura and K. Nakashima: "Set-valued functions and regularity," *Proc. of the 27th ISMVL*, pp. 89-94, 1997.

[7] I. G. Rosenberg, "Multiple-valued hyperstructures," *Proc. of the 28th ISMVL*, pp. 326-333, 1998.

SESSION IVB
ALGEBRA II
CHAIR: Reiner Hähnle



A Super Switch Algebra for Quantum Device based Systems

Gerhard W. Dueck*, Mou Hu†, and Blair Fraser*

Abstract

In this paper a multiple-valued algebra, super switch algebra, is formalized. The algebra is well suited for the design of quantum device based multiple-valued systems. A minimization algorithm based on this algebra is proposed. This new algebraic algorithm has some advantages over the existing map-based procedure and subfunction-based algorithm. Benchmark functions are minimized and the results are discussed. Further improvements to the algorithm are suggested.

1 Super Switch Algebra

With rapid increase in integrated circuit density, which will likely reach its limits in the near further, the search for alternative technologies is on. Among many emerging technologies, multiple-valued systems based on quantum devices are most promising [2, 8, 11, 13].

The objective of this paper is to propose an algebraic structure that can be used as a foundation for designing the next generation VLSI circuits. To achieve this goal the super pass gate model (renamed as super switch model in this paper) proposed by Deng et al. [3] is reviewed first. Then based on the switch-level algebra proposed by Hu [6], the super switch algebra is constructed. Properties of super switch algebra are studied in detail. Finally an algorithm based on super switch algebra for design of quantum device based multiple-valued digital systems is proposed. The algorithm has been implemented in C++. Benchmark functions are minimized to show the viability of the program. Careful analysis of some functions, allows us to propose improvements to the algorithm.

*Department of Mathematics, Statistics, and Computer Science, St. Francis Xavier University, Antigonish, N.S., B2G 2W5, CANADA, gdueck@stfx.ca. Research is partially supported by NSERC.

†Aepos Technologies, Adga Group, Ottawa, Ont. K1P 5G3, CANADA. Research was initiated while visiting St. Francis Xavier University as James Chair Professor.

2 Review of the Super Switch Model

After surveying many papers on quantum devices [2, 3, 6, 8, 9, 10, 11, 12, 13] we found that the super pass gate model [3] is one of the most promising models in this category. In this paper, we adopt super pass gate model and rename it as super switch, which will be used as a suggested building block of next generation VLSI multiple-valued systems. The following is a brief review of the super switch model.

A super switch is a device with a data terminal D, a control terminal C, an output terminal O, and a characteristic set S as illustrated in Fig. 1(a). The function of a super switch is depicted in Fig. 1(b). When the control signal C takes value from the characteristic set S, the switch is on and data D is passing to output O. Otherwise, the switch is off and data D cannot pass to output O.

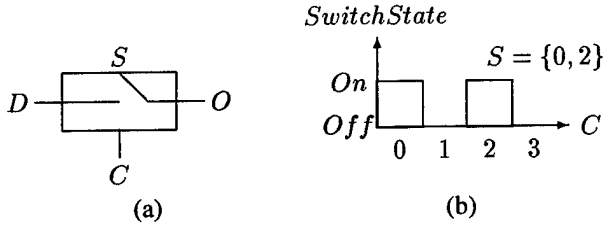


Figure 1. Super switch model.

Since the characteristic set S of a super switch is programmable, there will be r^2 different super switches in an r-valued system including two trivial switches, always on and always off. The possible implementation of a super switch using quantum device technology was proposed in [3].

3 Formalization of Super Switch Algebra

Deng et al. proposed two minimization algorithms for systems using super switches (superpass gates) in [4] and [3]. But the mathematical foundation of super switch systems is still incomplete. By carefully examining the concept of super switch, we found that the switch-level algebra proposed by Hu [6] can be used as a basis for developing an

0-7695-0161-3/99 $10.00 © 1999 IEEE

algebraic structure that will include the super switch model. Actually if we extend the logic value set used in switch-level algebra and replace positive and negative switches by super switches, a new algebraic structure can be formalized. We will call this new algebraic structure super switch algebra (SS algebra for short) which serves as a mathematical model for analyzing, designing, verifying, and testing quantum device based VLSI systems. This new algebra will be compatible with definitions proposed by Deng et al. but in a mathematically complete form.

3.1 Logic Value Sets of SS Algebra

We define a **functional logic value set** $R = \{0, 1, \ldots, r-1\}$, where $0, 1, \ldots, r-1$ are the logic values used in a sytem with radix r. Inputs and outputs of systems should take values in R. When we functionally specify a system, this set will be used.

The output O of a super switch may be in a high impedance state when the control varible is not in the set S. We must be able to incorporate this into the algebra. Hence, we define an **intermediate logic value set** $I = R \cup \{z\} = \{z, 0, 1, \ldots, r-1\}$ where z represents the high impedance state. Signals inside a system should take values in I. This set will be used during the design phase of a system.

The **total logic value set** $T = I \cup \{u\} = \{z, 0, 1, \ldots, r-1, u\}$, where u represents an undertermined logic value resulting from conflicting logic values. This set is used during design verification. In a properly designed system, u should never appear.

In the total logic value set T, we define a "logic strength greater than and equal to" relation. For every value in T, logic strength is assigned as follows.

- u's logic strength is the greatest.
- z's logic strength is the weakest.
- $0, 1, \ldots, r-1$ cannot compare with each other.

"Logic strength greater than or equal to" relation (denoted as $\geq$ relation) is a partial order in T. Thus, $< T, \geq >$ is a lattice as depicted in Figure 2.

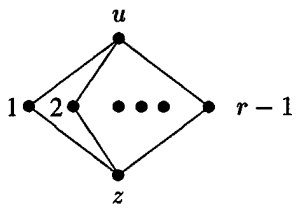


Figure 2. The lattice for the SS algebra.

Two operators, connection operator and super switch operator, will be defined for the super switch algebra. For the following definitions, x and y are constants or functions in T. The connection operator # is defined as:

$$x \# y = LUB(x, y) \tag{1}$$

where $LUB(x, y)$ is the least upper bound of x and y. In set I, the # operator is the same as the + operator proposed by Deng et al. in [3]. However, the # operator has a significant advantage over the + operator, because it allows for the connection with all values in T. The + operator disallowed the connection of two logic values for which the $\geq$ relation was not defined. By using the # operator, such a connection will yield a u indicating that this is not a valid output of the system, and thus it can be used during the design verification phase. The connection operator is a mathematical model of a connecting line in a digital circuit.

The *super switch operator* $\$^S$ is defined as

$$y \,\$^S\, x = \begin{cases} y & \text{if } x \in S \\ z & \text{if } x \notin S \text{ and } x \neq u \text{ and } y \neq u \\ u & \text{if } x = u \text{ or } y = u \end{cases} \tag{2}$$

where y is the pass variable; x is the control variable; $x, y \in T$; S is the characteristic set of the SS operator; $S \subseteq R$.

It is easy to see that the super switch operator serves as a mathematical model for the super switch described in Section 2. For the super switch shown in Fig. 1, we have $O = D \,\$^S\, C$. Deng et al. [3, 4] define a universal literal $< y > x^S$ whose function is similar to the function of our super switch operator (i.e. $y \,\$^S\, x$). We chose the new notation, because the super switch is a binary operation between two arbitrary functions. The former notation gives the impression that the control variable is limited to input variables only.

4 Properties of the Super Switch Algebra

In this section we will study the properties of super switch algebra in detail. All the properties will presented in the form of laws. In our discussion we adopt the following convention: in an expression, the priority of $\$^S$ operation is higher than the priority of # operation. Unless otherwise specified, A, B, C, x, x_1, x_2 are functions in T; a is a value in R; S, $S1$, $S2$ are subsets of R; and Φ is an empty set.

Law 1. Commutative law of # operation

$$A \# B = B \# A$$

Law 2. Associative law of # operation

$$A \#(B \# C) = (A \# B) \# C$$

Law 3. Idempotent law of # operation

$$A\#A = A$$

Law 4. Unit element and zero element of # operation

z is the unit element of # operation:

$$z\#A = A\#z = A$$

u is the zero element of # operation:

$$u\#A = A\#u = u$$

Law 5. Law of changing the order of two $\$^S$ operations

$$A\ \$^{S1}\ x_1\ \$^{S2}\ x_2 = A\ \$^{S2}\ x_2\ \$^{S1}\ x_1$$

Law 6. Associative law of $\$^S$ operation

$$(A\ \$^{S1}\ x_1)\ \$^{S2}\ x_2 = A\ \$^{S1}\ (x_1\ \$^{S2}\ x_2)$$

Law 7. Law of repeating a $\$^S$ operation

$$A\ \$^S\ x\ \$^S\ x = A\ \$^S\ x$$

Law 8. Zero elements of $\$^S$ operation

(1) u is the zero element of $\$^S$ operation in T:

$$A\ \$^S\ u = u\ \$^S\ A = u$$

(2) z is the zero element of $\$^S$ operation in I:

$$A\ \$^S\ z = z\ \$^S\ A = z$$

Law 9. Distributive law of $\$^S$ operation to # operation

$$(A\#B)\ \$^S\ x = (A\ \$^S\ x)\#(B\ \$^S\ x)$$

Law 10. Absorption law

If $x \in I$, we have

$$A\#(A\ \$^S\ x) = A$$

Law 11. The first law of merging two $\$^S$ operations

$$A\ \$^{S1}\ x\#A\ \$^{S2}\ x = A\ \$^{S1 \cup S2}\ x$$

Law 12. The second law of merging two $\$^S$ operations

$$A\ \$^{S1}\ x\ \$^{S2}\ x = A\ \$^{S1 \cap S2}\ x$$

Law 13. Law of deleting a $\$^R$ operation

If $x \in R$,

$$A\ \$^R\ x = A$$

Law 14. Substitution law of $\$^a$ operation

$$a\ \$^a\ x = x\ \$^a\ x$$

Law 15. Law of deleting an SS term

If $A, x \in I$,

$$A\ \$^\Phi\ x = z$$

All the above laws can be proven by using definitions of # and $\$^S$ operations, and/or by listing all possible combinations. A sample proof for Law 6 can be found in [7].

5 Use of Super Switch Algebra to Express Super Switch Networks

A super switch network is an extension of a pass transistor network. There are basically two types of super switch networks, i.e., series-parallel network (two levels) and tree-like network. We will only consider series-parallel network in this paper.

The following are some definitions of terms to be used to express a series-parallel super switch network. Some of them are similar to the definitions in [4]. An SS-term (also referred to as a product term) consists of a data function and a series of control variables connected with super switch operators (SS operators for short). We write the SS-term A as follows

$$A = y_a\ \$^{Sa1}\ x_1\ \$^{Sa2}\ x_2 \ldots\ \$^{San}\ x_n$$

A **standard SS-term** is one in which every variable appears exactly once. According to Law 13, any SS-term that does not include the variable x_j can be transformed into a standard SS-term by adding $\$^R\ x_j$. A **fundamental SS-term** is a standard SS-term with the following characteristics: (1) its data function is a constant; (2) the characteristic set of each SS operator contains only one logic value. It is easy to see that one fundamental SS-term corresponds to one row of a truth table of a logic function. We sometimes refer to a fundamental term as a **minterm**. It has been proven [4] that any n-variable r-valued function $F(X)$ can be expressed in the form of a parallel connection of fundamental SS-terms (also called sum of fundamental SS-terms) as

$$\sum_{a_1, a_2, \ldots, a_n = 0}^{r-1} F(a_1, a_2, \ldots, a_n)\ \$^{a1}\ x_1\ \$^{a2}\ x_2 \ldots\ \$^{an}\ x_n$$

where the symbol Σ represents the connection operators.

A sum of products is a series of one or more SS-terms connected with the connection operator #. By combining some fundamental SS-terms, any function can be represented by a sum of products. An SS-term A is said to

be an **implicant** of the function $f(X)$, if and only if for some assignments of $x, A(x) \neq z$ and for every assignment of x either $A(x) = f(x)$ or $A(x) = z$. An implicant A is said to be **prime** if there is no other implicant B such that $\forall x, A(x) \neq z \Rightarrow A(x) = B(x)$ and $\exists x, A(x) = z$ where $B(x) \in R$. It is easy to see, that any function can be expressed by a sum of products, where all SS-terms are prime implicants. An SS-term A **subsumes** another SS-term B if and only if A is an implicant of the function $F(X) = B$.

The consensus operations has been successfully used to find all prime implicants of a MVL function [1, 5]. The consensus operation of two SS-terms is defined to produce implicants that can be obtained from the two given implicants. An implicant generated by the consensus operation from given terms A and B, covers minterms from both A and B. Since the pass signal of an SS-term could be either a constant or a variable, the definition of consensus is more complex than the conventional one (such as the one described by Allen and Givone [1]). The consensus in the i^{th} dimension between the following two standard SS-terms, where $1 \leq i \leq n$

$$A = y_a \, \$^{Sa1} \, x_1 \, \$^{Sa2} \, x_2 \ldots \$^{San} \, x_n$$

$$B = y_b \, \$^{Sb1} \, x_1 \, \$^{Sb2} \, x_2 \ldots \$^{Sbn} \, x_n$$

is denoted as $A *^i B$ and can be defined as follows:

1. If both y_a and y_b are variables and $y_a = y_b$,

   $$A *^i B = y_a \, \$^{Sa1 \cap Sb1} \, x_1 \ldots \$^{Sai \cup Sbi} \, x_i \ldots \$^{San \cap Sbn} \, x_n$$

   Example 1. $x_1 \, \$^{0,1,2} \, x_1 \, \$^0 \, x_2 \, *^2 \, x_1 \, \$^{1,2,3} \, x_1 \, \$^2 \, x_2 = x_1 \, \$^{1,2} \, x_1 \, \$^{0,2} \, x_2$ (see Figure 3).

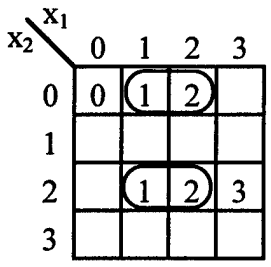


Figure 3. Illustration of case 1.

2. If both y_a and y_b are constants and $y_a = y_b$,

   $$A *^i B = y_a \, \$^{Sa1 \cap Sb1} \, x_1 \ldots \$^{Sai \cup Sbi} \, x_i \ldots \$^{San \cap Sbn} \, x_n$$

   Example 2. $1 \, \$^{0,1,2} \, x_1 \, \$^0 \, x_2 \, *^2 \, 1 \, \$^{1,2,3} \, x_1 \, \$^2 \, x_2 = 1 \, \$^{1,2} \, x_1 \, \$^{0,2} \, x_2$ (see Figure 4).

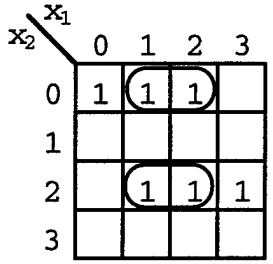


Figure 4. Illustration of case 2.

3. If both y_a and y_b are constants and $y_a \neq y_b$ and $y_a \in Sai$ and $y_b \in Sbi$,

   $$A *^i B = x_i \, \$^{Sa1 \cap Sb1} \, x_1 \ldots \$^{y_a \cup y_b} \, x_i \ldots \$^{San \cap Sbn} \, x_n$$

   Example 3. $[0 \, \$^{0,1,2,3} x_1 \, \$^0 \, x_2] *^2 [2 \, \$^{1,2} x_1 \, \$^{2,3} x_2] = x_2 \, \$^{1,2} \, x_1 \, \$^{0,2} \, x_2$ (see Figure 5).

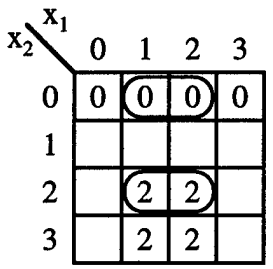


Figure 5. Illustration of case 3.

4. If both y_a and y_b are variables and $y_a \neq y_b$,

   Part 1. Let $y_a = x_j$, $y_b = x_k$ and $C = Saj \cap Sbk$. For each $c \in C$ let

   $$A' = c \, \$^{Sa1} \, x_1 \ldots \$^c \, x_j \ldots \$^{San} \, x_n$$

   $$B' = c \, \$^{Sb1} \, x_1 \ldots \$^c \, x_k \ldots \$^{Sbn} \, x_n$$

   The consensus of A' and B' can be treated as case 2.

   Part 2. Let $y_a = x_j$, $y_b = x_k$ and $C_a = Sai \cap Saj \cap Sbj$ $C_b = Sbi \cap Sbk \cap Sak$. For each $c_a \in C_a$ let

   $$A' = x_i \, \$^{Sa1} \, x_1 \ldots \$^{c_a} \, x_i \ldots \$^{c_a} \, x_j \ldots \$^{San} \, x_n$$

   For each $c_b \in C_b$ let

   $$B' = x_i \, \$^{Sb1} \, x_1 \ldots \$^{c_b} \, x_i \ldots \$^{c_b} \, x_k \ldots \$^{Sbn} \, x_n$$

   Find the consensus of each pair A', B' according to case 1.

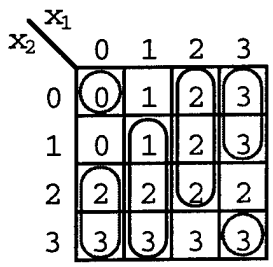


Figure 6. Illustration of case 4.

Example 4. $x_1 \ \$^{0,1,2,3} \ x_1 \ \$^{0,1} \ x_2 \ *^2 \ x_2\$^{0,1,2,3}x_1\$^{2,3}x_2 = 2\$^2x_1\$^{0,1,2}x_2 \#3\$^3x_1\$^{0,1,3}x_2 \# x_2 \$^0 \ x_1 \ \$^{0,2,3} \ x_2 \# x_2 \$^1 \ x_1 \ \$^{1,2,3} \ x_2$ (see Figure 6).

5. If y_a is a constant and y_b is a variable (swap A and B if necessary) and $y_a \in S_{ai}$ and $y_b = x_i$,

$$A *^i B = x_i \ \$^{Sa1 \cap Sb1} \ x_1 \ldots \$^{y_a \cup Sbi} \ x_i \ldots \$^{San \cap Sbn} \ x_n$$

Example 5. $0 \ \$^{0,3} \ x_1 \ \$^{0,1} \ x_2 \ *^2 \ x_2 \ \$^{0,1} \ x_1 \ \$^{2,3} \ x_2 = x_2 \ \$^0 \ x_1 \ \$^{0,2,3} \ x_2$ (see Figure 7).

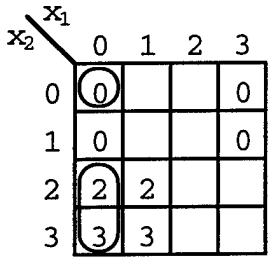


Figure 7. Illustration of case 5.

6. If y_a is a constant and y_b is a variable (swap A and B if necessary) and $y_a \in Sai$ and $y_b = x_j (j \neq i)$ and $\exists k$ $b_k \in (S_{bj} \cap S_{bi})$,

$$A *^i B = x_i \ \$^{Sa1 \cap Sb1} \ x_1 \ldots \$^{y_a \cup b_k} \ x_i \ldots \$^{Saj \cap b_k} \ x_j \ldots \$^{San \cap Sbn} \ x_n$$

If multiple k exist, there will be multiple results.

Example 6. $2 \ \$^{2,3} \ x_1 \ \$^{0,1} \ x_2 \ *^1 \ x_2 \ \$^{0,1} \ x_1 \ \$^{0,2} \ x_2 = x_1 \ \$^{0,2} \ x_1 \ \$^0 \ x_2$ (see Figure 8).

7. If y_a is a constant and y_b is a variable (swap A and B if necessary) and $y_a \in S_{bi}$ and $y_b = x_i$,

$$A *^i B = y_a \ \$^{Sa1 \cap Sb1} \ x_1 \ldots \$^{Sai \cup y_a} \ x_i \ldots \$^{San \cap Sbn} \ x_n$$

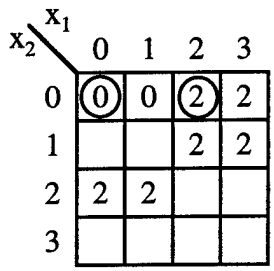


Figure 8. Illustration of case 6.

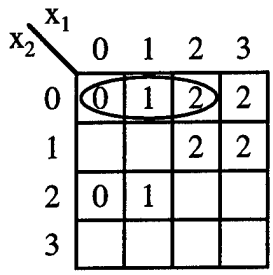


Figure 9. Illustration of case 7.

Example 7. $2 \ \$^{2,3} \ x_1 \ \$^{0,1} \ x_2 \ *^1 \ x_1 \ \$^{0,1} \ x_1 \ \$^{0,2} \ x_2 = x_1 \ \$^{0,1,2} \ x_1 \ \$^0 \ x_2$ (see Figure 9).

8. If y_a is a constant and y_b is a variable (swap A and B if necessary) and $y_a \in S_{bj}$ and $y_b = x_j$ $(j \neq i)$,

$$A *^i B = y_a \ \$^{Sa1 \cap Sb1} \ x_1 \ldots \$^{Sai \cup Sbi} \ x_i \ldots \$^{Saj \cap y_a} \ x_j \ldots \$^{San \cap Sbn} \ x_n$$

Example 8. $2 \ \$^{2,3} \ x_1 \ \$^{0,1,2} \ x_2 \ *^1 \ x_2 \ \$^{0,1} \ x_1 \ \$^{0,2} \ x_2 = x_2 \ \$^{0,1,2,3} \ x_1 \ \$^2 \ x_2$ (see Figure 10).

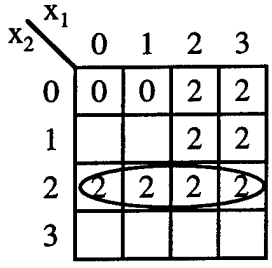


Figure 10. Illustration of case 8.

9. Otherwise,

$$A *^i B = z.$$

Note that the above definition of consensus is based on Law 11, Law 14, and Law 15. The consensus operation is defined with respect to two standard SS-terms. Terms in non-standard SS-forms must be transformed into standard forms before a consensus operation is applied.

In some cases the consensus operation will produce more than one term. For example, if the two SS-terms meet the criteria given in case 6, more than one k may exist and therefore several consensus terms may be generated. Furthermore, the two SS-terms may meet the requirements of more than one of the cases described above. For an illustration see the example below.

Example 9. Given two product terms $A = 0\$^0 x_1 \$^{0,1,2,3} x_2$ and $B = x_2\ \$^{1,2}\ x_1\ \$^{0,1,2,3}\ x_2$ we have $A *^1 B = x_1\ \$^{0,1}\ x_1\ \$^1\ x_2 \# x_1\ \$^{0,2}\ x_1\ \$^2\ x_2 \# 0\ \$^{0,1,2}\ x_1\ \$^0\ x_2$ (see Figure 11). The first two products are obtained from case 6 and the last one falls into case 8.

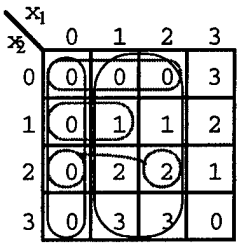


Figure 11. Map for a two-variable function

Lemma 1. *Let A and B be two SS-terms at distance 1 and let i be the dimension in which the corresponding S_i do not intersect. Then the only non-empty consensus exists in the i^{th} dimension.*

Lemma 2. *Let A and B be two SS-terms at distance greater than 1. Then the consensus between A and B is empty in all dimensions.*

Theorem 1. *Let A and B be implicants of the function $f(X)$ for which consensus in the i^{th} dimension exists. Then $C = A *^i B$ is also an implicant of $f(X)$.*

Proof. For each of the 9 cases of consensus it can be shown that for $\forall x$ such $C(x) \neq u$, $C(x) = A(x) \# B(x) = f(x)$. □

Theorem 2. *Let P be a sum of products representation of the function $f(X)$. Repeatedly remove from P all SS-terms, which subsume other SS-terms in P and add to P all SS-terms generated by consensus between SS-terms present in P until no new SS-terms are generated. P consists of just the prime implicants of function $f(X)$.*

Proof. Similar to [3]. □

6 Minimization

In this paper we only consider two-level minimization in which each term has the same cost. In other words we want to a "sum-of-products" with the minimum (or near minimum) number of terms. We implemented the classical two step minimization algorithm. First, all prime implicants are generated. Second, a minimal cover is found among the prime implicants. The algorithm for prime implicant generation follows directly from Theorem 2. Given a list of implicants, that cover the function, find the consensus terms for each adjacent pair (according to Lemma 2, we only need to consider terms at distance less than 2). Add it to the list if it is not subsumed by any of the terms in the list. Remove all subsumed terms. Repeat the procedure until no new terms are added. Finally, find an irredunant cover from the list of prime implicants.

Some results of our implementation in C++ are shown in Table 1. The cpu time was measured on a Pentium PC running a 166 MHz. All but two of the function in Table 1 are well known binary benchmarks. We converted each one to a 4-valued function by combining pairs of input variables and combining the first two outputs. "deng" is taken from [4] and "9var1" was constructed to show that the algorithm can easily minimize function with 9 input variables.

In the examples used to test the algorithm, the prime implicant generation required significantly more computation time than the finding of an irrdundant cover. It comes as no surprise that symmetric functions, such as *9sym*, are particularly difficult to minimize. This is due to the fact, that they often have a large number of prime implicants that can be generated in many different ways. In addition, an irredundant cover can be obtained in many ways.

The effort required to generate all prime implicants depends heavily on the initial input. As an illustration, we minimized the function *9sym* using different sets of terms as input to our program. The results are summarized in Table 2. *9sym* has 93 prime implicants. As the terms in the input are closer to prime implicants, the time to generate all prime implicants decreases dramatically. We are currently modifying our algorithm to preprocess the list of input terms to reduce the number of terms. Heuristics can be used for the preprocessing, since the iterative consensus will guarantee a list of prime implicants. Preliminary results are encouraging.

7 Conclusion

We have developed a multiple-valued algebra, super switch algebra, and a prime implicant generation algorithm based on it. This opens a new approach to the design of super switch networks, a one-pass algebraic approach. Compared with the map-based procedure presented in [4] and

Function	n	Terms	Time *sec*
deng	2	4	0.02
bw	3	4	0.04
rd53	3	2	0.05
xor5	3	2	0.01
5xp1	4	2	0.08
misex1	4	7	0.09
rd73	4	2	1.08
rd84	4	1	0.39
9sym	6	25	954.00
sao2	6	7	61.76
clip	6	4	748.00
9var1	9	5	0.10

Table 1. Computation time for some benchmark functions.

Terms	Time *sec*
93	6
121	9
140	13
163	17
177	33
181	36
186	66

Table 2. Minimization of 9sym using differnt input terms.

subfunction-based algorithm presented in [3], our method has several advantages:

- Due to its algebraic nature, it is more suitable to the synthesis of multiple-valued functions with many variables.
- It avoids decomposing the function into multiple subfunctions and thus avoids producing the same prime implicants in several of the subfunctions.
- The algorithm we proposed is easier to be implemented in computer software.
- It is potentially more useful for formal verification of design, test generation, and design for testability

References

[1] C. M. Allen and D. D. Givone. The Allen-Givone implementation oriented algebra. In D. C. Rine, editor, *Computer Science and Multiple-Valued Logic*, pages 268–288. North-Holland, 1984.

[2] T. Baba and T. Uemura. Multiple-junction surface transistor for multiple-valued logic circuits. In *Proceedings of the International Symposium on Multiple-Valued Logic*, pages 41–46, May 1997.

[3] X. Deng, T. Hanyu, and M. Kameyama. Quantum device model based super pass gate for multiple-valued digital systems. In *Proceedings of the International Symposium on Multiple-Valued Logic*, pages 92–97, May 1995.

[4] X. Deng, T. Hanyu, and M. Kameyama. Synthesis of multiple-valued logic networks based on super pass gates. In *Multiple-Valued Logic:An International Journal*, pages 161–183, 1996.

[5] G. W. Dueck and D. M. Miller. RCM-MVL: A recursive concensus MVL minimization algorithm. In *Proceedings of the International Symposium on Multiple-Valued Logic*, pages 136–143, May 1990.

[6] M. Hu. A multiple-valued algebra for modeling MOS VLSI circuits at switch-level. In *Proceedings of the International Symposium on Multiple-Valued Logic*, pages 329–336, May 1989.

[7] M. Hu and G. W. Dueck. A multiple-valued super switch algebra with applications to design of quantum device based systems. In *Fourth International Conference on Applications of Computer Systems*, pages 256–265. Technical University of Szczecin, Poland, Nov. 1997.

[8] H. C. Lin. Resonant tunneling diodes for multiple-valued digital applications. In *Proceedings of the International Symposium on Multiple-Valued Logic*, pages 188–195, May 1994.

[9] L. J. Micheel and H. L. Hartnagel. Interband RTDs with nanoelectronic HBT-LED structures for multiple-valued computation. In *Proceedings of the International Symposium on Multiple-Valued Logic*, pages 80–85, May 1996.

[10] H. Tang and H. C. Lin. Multiple-valued decoder based on resonat tunneling diodes in current tapping mode. In *Proceedings of the International Symposium on Multiple-Valued Logic*, pages 230–234, May 1996.

[11] T. Waho. Resonant tunneling transistor and its application to multiple-valued logic circuits. In *Proceedings of the International Symposium on Multiple-Valued Logic*, pages 130–138, May 1995.

[12] T. Waho, K. J. Chen, and M. Yamamoto. A literal gate using resonant-tunneling devices. In *Proceedings of the International Symposium on Multiple-Valued Logic*, pages 68–73, May 1996.

[13] T. Waho and M. Yamamoto. Application of resonant-tunneling quaternary quantitizer to ultrahigh-speed A/D converter. In *Proceedings of the International Symposium on Multiple-Valued Logic*, pages 35–40, May 1997.

Clarifying the Axioms of Kleene Algebra based on the Method of Indeterminate Coefficients

Tomoko Ninomiya
Information Science Center, Meiji Univ., Japan
ninomiya@kisc.meiji.ac.jp

Masao Mukaidono
Department of Computer Science, Meiji Univ., Japan
masao@cs.meiji.ac.jp

Abstract

After introducing the Method of Indeterminate Coefficients, by which we can derive all finite models satisfying a given set of axioms, we derive all models of 8 elements of Kleene algebra by using the method , and find out many examples of a set of independent and complete axioms of Kleene algebra by checking whether each axiom in Kleene algebra is independent from others or not based on the method.

1. Introduction

When fuzzy sets theory was introduced by L. A. Zadeh[1] to treat numerically ambiguity concepts concerning human thinking, linguistic meaning and so on, he used firstly three fundamental logic operations AND($\cap$), OR($\cup$) and NOT(~) and defined them as min, max and 1- on the closed interval [0,1], respectively. Since being introduced the logic operations AND, OR and NOT on [0,1] instead of two-valued {0,1}, the logic systems $<[0,1],\cap,\cup,\sim>$ was studied actively in the beginning under the name of fuzzy logic. Although the algebraic system corresponding to the fuzzy logic $<[0,1],\cap,\cup,\sim>$ was considered as De Morgan algebra, in 1981[2] one of the authors formulated firstly the fuzzy logic system $<[0,1],\cap,\cup,\sim>$ as Kleene algebra which is a stronger algebraic system than de Morgan algebra. Kleene algebra is an algebraic system which is a slightly weaker than Boolean algebra, that is, in Kleene algebra the complementary laws (the excluded middle law $\sim A \cup A = 1$ and the contradiction law $\sim A \cap A = 0$) do not hold but Kleene's laws $\sim A \cup A \geq \sim A \cap A$ hold, where Kleene's laws are weaker conditions than the complementary laws. Kleene algebra is a useful and interesting algebra for treating ambiguity or undefined or contradictory states, and it is well known that Kleene algebra corresponds essentially to three-valued logic. But it is not known what kind of finite models satisfy the Kleene algebra, so it is an interesting problem to find out the finite models satisfying Kleene algebra.

On the other hand the Method of Indeterminate Coefficients was developed by M. Goto[3]-[6]to determine the truth tables of undefined operators included in axioms of multiple-valued logic systems. Essentially it is considered that the Method of Indeterminate Coefficients is a method to find out finite models satisfying given axioms. Therefore the Method of Indeterminate Coefficients is applicable to find out finite models satisfying an algebraic system when the axioms are given. That is, it is a strong tool to proof an axiom is independent from other axioms (the designate axiom is not derived from other axioms), because we can apply the method to find out a model, which is a counter example, satisfies the set of other axioms but does not the designate axiom. In this paper after introducing the Method of Indeterminate Coefficients, we derive all models of 8 elements of Kleene algebra by using the method , and find out many examples of a set of independent and complete axioms of Kleene algebra by checking whether each axiom in Kleene algebra is independent from others or not based on the method.

2. Axioms of Kleene Algebra

Table 1. Representative equations valid in Kleene algebra

(1) the commutative law (a)	$X \cup Y = Y \cup X$
(1)'the commutative law (b)	$X \cap Y = Y \cap X$
(2) the associative law (a)	$X \cup (Y \cup Z) = (X \cup Y) \cup Z$
(2)'the associative law (b)	$X \cap (Y \cap Z) = (X \cap Y) \cap Z$
(3) the absorption law (a)	$X \cup (X \cap Y) = X$
(3)'the absorption law (b)	$X \cap (X \cup Y) = X$
(4) the distributive law (a)	$X \cap (Y \cup Z) = (X \cap Y) \cup (X \cap Z)$
(4)'the distributive law (b)	$X \cup (Y \cap Z) = (X \cup Y) \cap (X \cup Z)$
(5) the idempotent law (a)	$X \cup X = X$
(5)'the idempotent law (b)	$X \cap X = X$
(6) De Morgan's law (a)	$\sim (X \cup Y) = \sim X \cap \sim Y$
(6)'De Morgan's law (b)	$\sim (X \cap Y) = \sim X \cup \sim Y$
(7) the double negation law	$\sim (\sim X) = X$
(8) the least element (a)	$0 \cup X = X$
(8)'the least element (b)	$0 \cap X = 0$
(9) the greatest element (a)	$1 \cup X = 1$
(9)'the greatest element (b)	$1 \cap X = X$
(10) the Kleene's law (a)	$(X \cup \sim X) \cup (Y \cup \sim Y) = Y \cup \sim Y$
(10)'the Kleene's law (b)	$(X \cap \sim X) \cap (Y \cup \sim Y) = X \cap \sim X$

An algebraic system $<K,\cap,\cup,\sim,1,0>$ is Kleene algebra when a set K has at least two elements 1 and 0

0-7695-0161-3/99 $10.00 © 1999 IEEE

and the operators $\cap$, $\cup$ and ~ satisfy all equations of Table 1 for any elements X,Y,Z of K. We call these equations as axioms of Kleene algebra[2],[7], but these equations are not independent each other.

3. The Method of Indeterminate Coefficients

3.1. Truth-Values and Symbols

In this paper, we use the same following definitions as in the preceding papers[2]-[6].

3.1.1. Truth-Values

X, Y, Z are logical variables, which can take any elements on a set of K of truth values, and also take the truth value of "Greatest"("Truth") 1 or "Least"("False") 0 in case of two-valued logical algebraic system. Generally, in case of n_0-valued logical algebraic system, X,Y,Z take one of the sets of values K={1,2,$\cdots$, n_0} where 1, n_0 are designated as "Greatest"("Truth"), "Least"("False") and others are as "Intermediate", respectively.

"X takes the value x" is represented by

$X=x$ (3.1.1.1)

and simply by the two-valued expression X^x. Then

$X^x \equiv (X = x)$ (3.1.1.2)

,where symbol $\equiv$ is referred to in the next (1) in Section 3.1.2.

It means that the truth-value of (3.1.1.2) is 1(Truth) if X takes the value x and it is 0(False) if X doesn't take the value x. And then, the following equations are obtained.

about two-valued logic, $X^1 \vee X^0 \equiv 1$,

where $X^1 \cdot X^0 \equiv 0$ (3.1.1.3)

about three-valued logic, $X^1 \vee X^2 \vee X^3 \equiv 1$,

where $X^1 \cdot X^2 \equiv X^1 \cdot X^3 \equiv X^2 \cdot X^3 \equiv 0$ (3.1.1.4)

about n_0-valued logic $X^1 \vee X^2 \vee \cdots \vee X^{n_0} \equiv 1$,

where $X^i \cdot X^j \equiv 0$ for $\forall i, j; i \neq j$ (3.1.1.5)

Of course, the values 1 and 0 in the right side of $\equiv$ mean Truth and False of two-valued logic, where the symbols $\vee, \cdot$ means classical two-valued logic operators which is defined in the next (1) in Section 3.1.2.

3.1.2 Symbols

Operators which are used for discussion (in a meta-logic) are of course for two-valued logic and operators which are under research (in an object-logic) are for multiple-valued logic in this case and these two operators must be clearly distinguished. That is, in case both logic are treated together, each operator for both logic is described as follows.

(1) Operators for Classical Two-valued Logic

Ordinarily used operators for classical two-valued logic are as shown in the followings, where $X, Y \in \{0,1\}$.

Negation $\overline{X} = 1 - X$,

Disjunction $X \vee Y = \max(X, Y)$

Conjunction $X \wedge Y$ *or* $X \cdot Y$ (if not confused, it can be omitted) $= \min(X, Y)$,

Implication $X \rightarrow Y = \overline{X} \vee Y$,

Equivalence $X \leftrightarrow Y = (X \rightarrow Y) \cdot (Y \rightarrow X)$,

Identical Equivalence $X \equiv Y$, which means that "X and Y are equivalent for every value of the set".

Equal $X=Y$, an operator which represents equation of algebra.

(2) Operators for Multiple-valued Logic

$\cap$, $\cup$ and ~ are operators on multiple-valued logic and, those are treated as undefined or unknown operators at first and the truth tables will be determined through the Method of Indeterminate coefficients.

3.2.The Method of Indeterminate Coefficients

3.2.1.Representation of Undefined Operators

$\cap$, $\cup$ are defined as two variable operators and ~ is as a single variable operator. Assuming that the formers are the unknown logical functions for two variables and the latter is the unknown logical function for one variable, names of functions are represented as *DIS, CON, NEG*. In this case, using the indeterminate coefficients, S^j_{mn}, C^j_{mn}, N^j_m which take two-value 0 or 1, those functions are expressed as follows.

$$(X \cup Y)^j = DIS(X,Y)^j = \bigvee_{m=1}^{n_0} \bigvee_{n=1}^{n_0} S^j_{mn} \cdot X^m \cdot Y^n . \quad (3.2.1.1)$$

$$(X \cap Y)^j = CON(X,Y)^j = \bigvee_{m=1}^{n_0} \bigvee_{n=1}^{n_0} C^j_{mn} \cdot X^m \cdot Y^n . \quad (3.2.1.2)$$

$$(\sim X)^j = NEG(X)^j = \bigvee_{m=1}^{n_0} N^j_m \cdot X^m . \quad (3.2.1.3)$$

, where S^j_{mn} is 1 if *DIS*(m, n)=j is true and 0 if *DIS*(m, n)=j is false. It is similar for C^j_{mn} *and* N^j_m, respectively.

(Example)

Table2(a)F(X,Y)

	1	2	3
1	3	2	3
2	2	1	3
3	1	1	2

Table2(b)F(X,Y)^1

	1	2	3
1	0	0	0
2	0	1	0
3	1	1	0

Table2(c)F(X,Y)^2

	1	2	3
1	0	1	0
2	1	0	0
3	0	0	1

Table 2(d) F(X,Y)^3

	1	2	3
1	1	0	1
2	0	0	1
3	0	0	0

Table 2(e) F(X,Y)

	1	2	3
1	F_{11}	F_{21}	F_{31}
2	F_{12}	F_{22}	F_{32}
3	F_{13}	F_{23}	F_{33}

The three-valued logical function $F(X,Y)$ of two variables in Table 2(a) is represented by $F(X,Y)^1$, $F(X,Y)^2$ and $F(X,Y)^3$ in Table 2(b)~(d), respectively. Only 0 and 1

appear in those tables. In this case, for example, the following three equations are obtained.

$$F(X,Y)^1 = X^1 \cdot Y^3 \vee X^2 \cdot Y^2 \vee X^2 \cdot Y^3$$
$$F(X,Y)^2 = X^1 \cdot Y^2 \vee X^2 \cdot Y^1 \vee X^3 \cdot Y^3 \quad (3.2.1.4)$$
$$F(X,Y)^3 = X^1 \cdot Y^1 \vee X^3 \cdot Y^1 \vee X^3 \cdot Y^2$$

And then, the following two conditions are concluded in case we use F_{mn}^j as the value of $X^m \cdot Y^n$ in $F(X,Y)^j$ $(1 \le j \le 3)$.

$$F_{mn}^1 \vee F_{mn}^2 \vee F_{mn}^3 = 1,$$
$$F_{mn}^1 \cdot F_{mn}^2 = F_{mn}^2 \cdot F_{mn}^3 = F_{mn}^1 \cdot F_{mn}^3 = 0, \ (1 \le m,n \le 3) \quad (3.2.1.5)$$

Generally, for example, when the three-valued logical function $F(X,Y)$ with two unknowns is represented by Table 2(e) where $F_{mn} \in \{1,2,3\}$ then the following expression is obtained.

$$F(X,Y)^j = F_{11}^j \cdot X^1 \cdot Y^1 \vee F_{12}^j \cdot X^1 \cdot Y^2 \vee \cdots \vee F_{33}^j \cdot X^3 \cdot Y^3 \quad (j = 1,2,3) \quad (3.2.1.6)$$

, where F_{mn}^j takes only one of the values 0 or 1, and

$$\bigvee_{j=1}^{3} F_{mn}^j = 1, \quad F_{mn}^j \cdot F_{mn}^k = 0 \ (j \ne k) \quad (3.2.1.7)$$

That is, when $F(X,Y)$ is an unknown logical function and the value of F_{mn}^j will be determined as 0 or 1, $F(X,Y)$ will be determined. The Method of Indeterminate Coefficient is a method which determinates the value of F_{mn}^j on condition such that a given set of equations is satisfied. In this case, for example, if $F_{mn}^1 = F_{mn}^2 = 0$ $F_{mn}^1 = F_{mn}^2 = 0$ is determined, then $F_{mn}^3 = 1$, or if $F_{mn}^3 = 1$ is determined, then $F_{mn}^1 = F_{mn}^2 = 0$.

3.2.2 Expression of the Axioms of Kleene Algebra using the Method of Indeterminate Coefficients

Assuming that each equation of Table 1 is treated as tautology, equation of Table 1, (1) the commutative law(a) is, for example, expressed as follows.

$$[X \cup Y = Y \cup X]^j \equiv 1$$

From this equation, clearly, $X \cup Y$ and $Y \cup X$ take the same value j for any truth value of X and Y. Then, $DIS(X,Y)^j$ and $DIS(Y,X)^j$ are satisfied simultaneously, where "j" takes any value of the values, $1,2,\cdots, n_0$. We obtain;

$$^{\exists} j \in \{1,2,\cdots,n_0\}; DIS(X,Y)^j \cdot DIS(Y,X)^j \equiv 1. \quad (3.2.2.2)$$

, where "j" takes only one value of $\{1,2,\ldots n_0\}$, simultaneously, then the equation (3.2.2.2) is expressed as follows.

$$\bigvee_{j=1}^{n_0} DIS(X,Y)^j \cdot DIS(Y,X)^j \equiv 1$$
$$\text{,but } {}^{\forall} i \ne j; DIS^i \cdot DIS^j \equiv 0 \quad (3.2.2.3)$$

Replacing DIS with S_{mn}^j ((3.2.1.1)) in above equation, the following equation can be obtained.

$$\bigvee_{j=1}^{n_0} S_{xy}^j \cdot S_{yx}^j \cdot X^x \cdot Y^y \equiv 1, \quad (3.2.2.4)$$

This equation must always be satisfied when logical variables X and Y take any truth values x and y. And then, the equation (3.2.2.4) is transformed into the equations (3.2.2.5) which is equivalent to (3.2.2.6).

$$\bigwedge_{x,y=1}^{n_0} \bigvee_{j=1}^{n_0} S_{xy}^j \cdot S_{yx}^j \cdot X^x \cdot Y^y \equiv 1, \quad (3.2.2.5)$$

$$\bigwedge_{x,y=1}^{n_0} \bigvee_{j=1}^{n_0} S_{xy}^j \cdot S_{yx}^j \equiv 1. \quad (3.2.2.6)$$

Therefore, clearly, the truth table about $\cup$, which satisfies the equation (1) of Table 1, can be determined from the undefined two-valued coefficients S_{xy}^j by solving the equation (3.2.2.6).

Simultaneously, all equations of Table 1 are transformed into equations of Table 3 using the undefined two-valued coefficients S_{mn}^j, C_{mn}^j, N_m^j in the equations (3.2.1.1)~(3.2.1.3), where x, y, z mean the value of independent variables X, Y, Z and j, k, l, m mean those of terms DIS, CON, NEG.

3.3 Algorithm for the Method of Indeterminate Coefficients

In this algorithm, at first we put "2", which means indeterminate, into all coefficients S_{mn}^j, C_{mn}^j *and* N_m^j. Next, we determinate the indeterminate coefficients "2" to determinate value "0" or "1" in order to satisfy each equation, one after another. That is, at first, in the n_0-valued logic, we determinate the coefficients in order to satisfy first equation about all sets of values for (x, y, z); from (1,1,1) to (n_0, n_0, n_0) in case of three variables, where some coefficients are not determined, i.e. those values are still "2". And then, we determinate those indeterminate coefficients, which are "2", by next equation. In this time, when a solution of the new equation shows a coefficient "1(0)", which was determined to "0(1)" in the previous equations, such a solution should be cancelled as a contradiction.

Similarly, the above process should be repeated for every set of values for x, y, z about all equation of a set of axioms. Finally, any solutions obtained during such process satisfy all equation of the axioms, which are models satisfying the given all axioms. Let us call the solutions during the process "Provisional solutions". Such algorithm is described in detail as follows.

3.3.1.The Program Based on Wide-spread Algorithm

We apply a wide-spread algorithm to each equation,

one after another, in order to determinate the coefficients by solving the simultaneous logical equations.

The procedure to apply such algorithm is as follows.

Procedure 1. Solving the first logical eqaution.

Procedure 1-1. We will get all "Particular solutions" which satisfy the first equation for (x, y, z)=(1, 1, 1).

Procedure 1-2. We will get new particular solutions about the first equation for next values of (x, y, z) and get new provisional solutions.

This process is repeated from (x, y, z)=(1, 1, 2) to (n_0, n_0, n_0).

Table 3. Equation of Table 1 transformed by undefined coefficients

(1) $\bigwedge_{x,y=1}^{n_0} \bigvee_{j=1}^{n_0} S_{xy}^{j} \cdot S_{yx}^{j} \equiv 1$	(3.2.2.7(1))
(1)' $\bigwedge_{x,y=1}^{n_0} \bigvee_{j=1}^{n_0} C_{xy}^{j} \cdot C_{yx}^{j} \equiv 1$	(3.2.2.7(1)')
(2) $\bigwedge_{x,y,z=1}^{n_0} \bigvee_{j,k,l=1}^{n_0} S_{xk}^{j} \cdot S_{yz}^{k} \cdot S_{lz}^{j} \cdot S_{xy}^{l} \equiv 1$	(3.2.2.7(2))
(2)' $\bigwedge_{x,y,z=1}^{n_0} \bigvee_{j,k,l=1}^{n_0} C_{xk}^{j} \cdot C_{yz}^{k} \cdot C_{lz}^{j} \cdot C_{xy}^{l} \equiv 1$	(3.2.2.7(2)')
(3) $\bigwedge_{x,y=1}^{n_0} \bigvee_{j=1}^{n_0} S_{xj}^{x} \cdot C_{xy}^{j} \equiv 1$	(3.2.2.7(3))
(3)' $\bigwedge_{x,y=1}^{n_0} \bigvee_{j=1}^{n_0} C_{xj}^{x} \cdot S_{xy}^{j} \equiv 1$	(3.2.2.7(3)')
(4) $\bigwedge_{x,y,z=1}^{n_0} \bigvee_{j,k,l,m=1}^{n_0} C_{xk}^{j} \cdot S_{yz}^{k} \cdot S_{lm}^{j} \cdot C_{xy}^{l} \cdot C_{xz}^{m} \equiv 1$	(3.2.2.7(4))
(4)' $\bigwedge_{x,y,z=1}^{n_0} \bigvee_{j,k,l,m=1}^{n_0} C_{xk}^{j} \cdot S_{yz}^{k} \cdot S_{lm}^{j} \cdot C_{xy}^{l} \cdot C_{xz}^{m} \equiv 1$	(3.2.2.7(4)')
(5) $\bigwedge_{x=1}^{n_0} S_{xx}^{x} \equiv 1$	(3.2.2.7(5))
(5)' $\bigwedge_{x=1}^{n_0} C_{xx}^{x} \equiv 1$	(3.2.2.7(5)')
(6) $\bigwedge_{x,y=1}^{n_0} \bigvee_{j,k,l,m=1}^{n_0} N_{k}^{j} \cdot S_{xy}^{k} \cdot C_{lm}^{j} \cdot N_{x}^{l} \cdot N_{y}^{m} \equiv 1$	(3.2.2.7(6))
(6)' $\bigwedge_{x,y=1}^{n_0} \bigvee_{j,k,l,m=1}^{n_0} N_{k}^{j} \cdot C_{xy}^{k} \cdot S_{lm}^{j} \cdot N_{x}^{l} \cdot N_{y}^{m} \equiv 1$	(3.2.2.7(6)')
(7) $\bigwedge_{x=1}^{n_0} \bigvee_{j=1}^{n_0} N_{j}^{x} \cdot N_{x}^{j} \equiv 1$	(3.2.2.7(7))
(8) $\bigwedge_{x=1}^{n_0} S_{mx}^{x} \equiv 1$	(3.2.2.7(8))
(8)' $\bigwedge_{x=1}^{n_0} C_{mx}^{x} \equiv 1$	(3.2.2.7(8)')
(9) $\bigwedge_{x=1}^{n_0} S_{1x}^{1} \equiv 1$	(3.2.2.7(9))
(9)' $\bigwedge_{x=1}^{n_0} C_{1x}^{1} \equiv 1$	(3.2.2.7(9)')
(10) $\bigwedge_{x,y=1}^{n_0} \bigvee_{j,k,l,m=1}^{n_0} S_{kj}^{l} \cdot C_{xl}^{k} \cdot N_{x}^{l} \cdot S_{ym}^{l} \cdot N_{y}^{m} \equiv 1$	(3.2.2.7(10))
(10)' $\bigwedge_{x,y=1}^{n_0} \bigvee_{j,k,l,m=1}^{n_0} C_{kj}^{l} \cdot C_{xl}^{k} \cdot N_{x}^{l} \cdot S_{ym}^{k} \cdot N_{y}^{m} \equiv 1$	(3.2.2.7(10)')

Procedure 2. Solving the second logical equation.

And then, we will get provisional solutions about the second equation of Table 3 to repeat the procedure 1-1. Those solutions become the general solutions about two equations to be treated.

Procedure 3. We will get general solutions about the set of axioms of Kleene algebra.

Repeating above the process to solve all equations of Table 3, we will get all solutions about the set of axioms of Kleene algebra, where each solution corresponds to a model of kleene algebra.

3.3.2 The Program Based on Depth-first Algorithm

In the program based on a depth-first algorithm, at first, only one solution which satisfy all equations of the set of axioms is derived. For that purpose, the coefficients *S*, *C* and *N* in each equation of the set of axioms must be solved in order to satisfy each equation for every set of values about x,y,z and some set of values about j,k,l,m. By storing the each set of values about j,k,l,m for all sets of values about x,y,z in each equation of an set of axioms, another solutions except the first one is derived efficiently. This method have the following two merits.

(1) About the size of the program

The size of this program based on this algorithm is smaller than that based on wide-spread algorithm, because only one provisional solution is treated always. And it is not necessary accessing any file on the disk memory. Therefore, in the many-valued algebraic system which can't be treated by the program based on wide-spread algorithm for limitation of the quantity of the disk memory, we can derive all solutions without considering the size of program and the disk memory.

(2) About deriving the solutions

If only one solution is needed, we can derive more fast by this program than by the program based on wide-spread method.

For example, we can obtain all models of a Kleene algebra having eight elements by using the program on Depth-first Algorithm. The Figure 2 shows all that models. In the next section we will show an application of this program of wide-spread method.

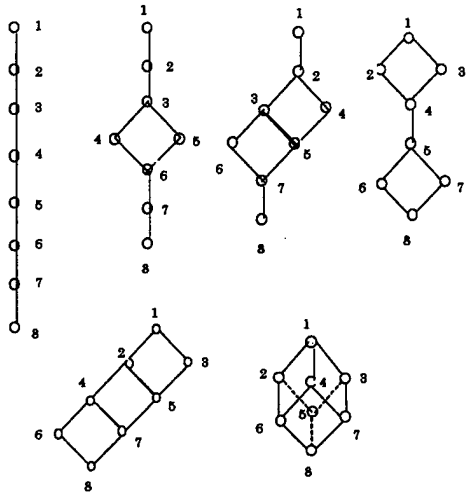


Figure 1. The Models of Kleene Algebra, n=8

4. Independent and Complete Axioms for Kleene Algebra

4.1. The Independence of Each Axioms for Kleene Algebra

The independence of each axiom for an algebraic system can be examined by applying the Method of Indeterminate Coefficients, in which the program is based on wide-spread method, to the equations of its set of axioms. For example, if we can find a new finite model of Kleene algebra, during the treatment of the last axiom, the independence of the last axiom from the set of previous axioms is proved automatically because the derived new model is a counter example showing the last axiom is independent from the set of previous axioms, and the last axiom become a candidate for complete axioms. Therefore, if we would like to decide whether an axiom is independent or not, it is sufficient to treat its equation at last in a set of axioms in the program of wide-spread method. For example, Table 4-8 show counter examples showing each axiom is independent from set of other axioms, respectively. Then, it was proved that one of the commutative law (a) or (b), one of the distributive law (a) or (b), one of the De Morgan's law (a) or (b), the double negation law, and one of the Kleene's law (a) or (b) are independent from others, respectively.

Table 4: An example which indicates that the commutative law is independent.

	1	2	3	4
1	1	1	1	1
2	1	2	2	2
3	1	2	3	4
4	1	2	3	4

$X \cup Y$

	1	2	3	4
1	1	2	3	4
2	1	2	3	4
3	3	3	3	3
4	4	4	4	4

$X \cap Y$

X	~X
1	4
2	3
3	2
4	1

Table 5: An example which indicates that the distributive law is independent

	1	2	3	4	5
1	1	1	1	1	1
2	1	2	1	1	2
3	1	1	3	1	3
4	1	1	1	4	4
5	1	2	3	4	5

$X \cup Y$

	1	2	3	4	5
1	1	2	3	4	5
2	2	2	5	5	5
3	3	5	3	5	5
4	4	5	5	4	5
5	5	5	5	5	5

$X \cap Y$

X	~X
1	5
2	4
3	3
4	2
5	1

Table 6. An example which indicates that the De Morgan's law is independent.

	1	2	3	4
1	1	1	1	1
2	1	2	3	2
3	1	3	3	3
4	1	2	3	4

$X \cup Y$

	1	2	3	4
1	1	2	3	4
2	2	2	3	2
3	3	2	3	4
4	4	4	4	4

$X \cap Y$

X	~X
1	2
2	1
3	4
4	3

Table 7. An example which indicates that the double negation law is independent.

	1	2	3
1	1	1	1
2	1	2	2
3	1	2	3

$X \cup Y$

	1	2	3
1	1	2	3
2	2	2	3
3	3	3	3

$X \cap Y$

X	~X
1	1
2	1
3	1

Table 8. An example which indicates that the Kleene's law is independent.

	1	2	3	4
1	1	1	1	1
2	1	2	1	2
3	1	1	3	3
4	1	2	3	4

$X \cup Y$

	1	2	3	4
1	1	2	3	4
2	2	2	4	4
3	3	4	3	4
4	4	4	4	4

$X \cap Y$

X	~X
1	4
2	2
3	3
4	1

In case of finite Kleene algebra, 1 and 0 are the greatest element and the least element by expressing the following equations, respectively.

$1 = X_1 \cup X_2 \cup \cdots \cup X_n, 0 = X_1 \cap X_2 \cap \cdots \cap X_n,$
for $X_1, X_2, \cdots, X_n \in$ the set of all elements of finite Kleene algebra

Figure 2.

Clearly, in case of finite Kleene algebra, the least element and the greatest element are not independent. But, in case of infinite Kleene algebra, for example, a model expressed in Figure 2 satisfies all axioms except the least element and the greatest element. Then, at least, one of the least element (a) or (b) or the greatest element (a) or (b) is independent from another axioms in Table 1.

The associative law (a) and (b), the absorption law (a) and (b), and idempotent law (a) and (b) are derived from another axioms in Table 1[2]. Then, those axioms are not independent.

4.2. Independent and Complete Axioms for a Kleene Algebra

We can find all independent axioms in Table 1 by applying the Method of Indeterminate Coefficients (Table 9). They are candidates for independent and complete axioms of Kleene algebra. Here, we try to derive examples of independent and complete axioms of Kleene algebra from the set of candidate axioms.

All sets of axioms included one of the least law (b) or the greatest law (a) but not included the least law (a) and the greatest law (b) are not complete. For example, a operator table (Table 10) satisfies the set of axioms in Table 11, but is not a model of the Kleene algebra. Therefore, the set of axioms in Table 11 is not complete.

If a candidate of independent and complete set of axioms includes the least element (a), then the greatest element (b) can be derived from the definition ~1=0, the De Morgan's law (b) and the double negation law.

Table 9.Independent axioms in table 1

{1} one of the the commutative law (a) or the commutative law (b)
{2}one of the the distributive law (a) or the distributive law (b)
{3}one of the De Morgan's law (a) or De Morgan's law (b)
{4}the double negation law
{5}one of the the least element (a) or the greatest element (b)
{6}one of the the Kleene's law (a) or the Kleene's law (b)

Table 10. An example of an operator table, which satisfies a set of axioms in Table 11.

$X \cup Y$	1	2	3
1	1	3	3
2	3	2	3
3	3	3	3

$X \cap Y$	1	2	3
1	1	3	3
2	3	2	3
3	3	3	3

X	~X
1	2
2	1
3	3

Similarly, the greatest elsement(b) can be derived from the definition ~0=1, the De Morgan's law and the double negation law. Then, only one of the least element (a) or the greatest element (b) is independent. From the results 4.1, all sets of axioms in Table 9 are independent from another axioms in Table 1. Therefore all independent and complete sets of axioms for Kleene algebra must include those six axioms in Table 9. In fact, it was proved[2] that the set of axioms of Table 12 is an independent and complete set of axioms of Kleene algebra. It means that all axioms in Table 1 can be derived from the set of axioms in Table 12. In fact, by using the results of reference (2), although here the proofs are omitted because of the space limit, we can prove that the 32 sets of axioms composed from one of the axioms 1-6 in Table 9 are independent and complete sets of axioms of the Kleene algebra.

Table 11. A set of axioms not satisfied Kleene algebra

{1} the commutative law (a)	$X \cup Y = Y \cup X$
{2} the distributive law (a)	$X \cap (Y \cup Z) = (X \cap Y) \cup (X \cap Z)$
{3} De Morgan's law (a)	$\sim (X \cup Y) = \sim X \cap \sim Y$
{4} the double negation law	$\sim (\sim X) = X$
{5} the least element (b)	$0 \cap X = 0$
{6} the Kleene's law (a)	$(X \cup \sim X) \cup (Y \cup \sim Y) = Y \cup \sim Y$

Table 12. A independent and complete set of axioms of Kleene algebra[2]

{1} the commutative law (a)	$X \cup Y = Y \cup X$
{2} the distributive law (a)	$X \cap (Y \cup Z) = (X \cap Y) \cup (X \cap Z)$
{3} De Morgan's law (a)	$\sim (X \cup Y) = \sim X \cap \sim Y$
{4} the double negation law	$\sim (\sim X) = X$
{5} the least element (a)	$0 \cup X = X$
{6} the Kleene's law (a)	$(X \cup \sim X) \cup (Y \cup \sim Y) = Y \cup \sim Y$

5. Conclusion

It is shown that the Method of Indeterminate Coefficients is very useful to find out finite models, and therefore to find independent and complete sets of axioms. Using a set of axioms of Kleene algebra as an example, we can show 32 independent and complete sets of axioms of Kleene algebra in total by the Method of Indeterminate Coefficients.

In the near future, we would like to apply this method to clarification of the fundamental properties of such another algebra.

References

[1] L. A. Zadeh, "Fuzzy Sets", Information and Control, Vol. 8, 1965.

[2] M. Mukaidono, "A Set of Independent and Complete Axioma for a Fuzzy Algebra(Kleene Algebra)", Proceedings of the 11th International Symposium on Multiple-Valued Logic, IEEE, May, 1981.

[3] M. Goto, S. Kao, T. Ninomiya, "Determination of Many-Valued Truth Tables for Undefined Operators in Axioms by a Computer and their Applications", Proceedings of the 7th International Symposium on Multiple-Valued Logic, IEEE, May, 1977.

[4] M. Goto, S. Kao, T. Ninomiya, "Axiomaization of Kleene's Three-Valued Logic by a Computer", Proceedings of the 9th International Symposium on Multiple-Valued Logic, IEEE, May, 1979.

[5] M. Goto, S. Kao, T. Ninomiya, "Axiomaization of Bochvar's Three-Valued Logic by a Computer", Proceedings of the 11th International Symposium on Multiple-Valued Logic, IEEE, May, 1981.

[6] M. Goto, S. Kao, T. Ninomiya, "Synthesis of Axiom Systems for the Three-Valued Predicate Logic by Means of Special Four Logic", Proceedings of the 13th International Symposium on Multiple-Valued Logic, IEEE, May, 1984.

[7] R. Balbes, P. Dwinger, "Distributive Lattices", University of Missouri Press, 1974.

The Number of Cascade Functions

Grant Pogosyan
International Christian University
Division of Natural Sciences
Mitaka, Tokyo 181-8585, Japan
grant@icu.ac.jp

Abstract

Cascade function is a Boolean function which can be implemented by a so-called cascade network. An n input cascade network is a circuit built with $n-1$ two-input-one-output gates (i.e., dyadic operations) such that at least one input of each gate is a network input. By arranging the inputs in a proper order these networks can be presented in a "cascade" shape, which is the origin of the name.

Cascade networks, and thus cascade functions have many interesting properties. Although the portion of cascades among all logic functions is small (the ratio approaches zero with growth of the number of variables), their remarkable properties and the practical significance of such networks has attracted many researchers in the fields of switching functions and logic design.

There are several papers published that focus on the enumeration problem of cascade functions and networks. Asymptotic expression and a recurrence relation have been found for the number of all cascades ([12]), as well as for some subclasses (see e.g. [13, 15]). However, to the author's knowledge, until now no closed formula has been discovered which explicitly counts such functions.

This paper presents an explicit formula for the number of all n-variable cascade functions.

1. Preliminaries

Let $B = \{0, 1\}$. An n-variable Boolean function is a map

$$f(x_1, \ldots, x_n) : B \to B^n.$$

Denote by $\Omega(n)$ the set of all n-variable Boolean functions. Put $\Omega = \sum_{i=1}^{\infty} \Omega(n)$.

The variable x_i of the function $f(x_1, \ldots, x_n)$ is called *essential*, if there exist values $a_1, \ldots, a_{n-1}, a_{n+1}, \ldots, a_n$:

$$f(a_1, \ldots, a_{n-1}, 0, a_{n+1}, \ldots, a_n) \neq f(a_1, \ldots, a_{n-1}, 1, a_{n+1}, \ldots, a_n).$$

Function $f \in \Omega(n)$ is called *essential function* if all its variables are essential. By $\Omega^e(n)$ we denote the set of all n-variable essential functions.

The number $E(n) = |\Omega^e(n)|$ of all n-variable essential functions are easily computed by –

$$E(n) = \sum_{i=0}^{n} (-1)^i \binom{n}{i} 2^{2^{(n-i)}}. \qquad (1)$$

The notion of *formula* over the subset $A \subseteq \Omega$ is inductively defined by superposition of functions from A, as well as the operations of identification and permutation of the variables. Any functions $f \in \Omega(n)$ can be expressed by a formula over the set $\Omega(2)$, which means the latter is *complete* in Ω.

Any function $f(x_1, \ldots, x_n) \in \Omega(n)$ has infinitely many formal expressions, also called decompositions over any complete domain A. An important problem for logic functions is to find an optimal expression over a given domain. In classical Logic Design $\Omega(2)$ and its certain complete subsets are most widely chosen domains of basic operations, i.e., considering combinational circuits that are built using two-input-one-output gates.

The general problem of optimal decomposition for arbitrary function is known to be hard. However, the study of certain classes of "easily decomposable" functions is important for a better understanding of the general optimization problem. Also, networks and circuits that realize these functions have remarkable properties, such as simplicity of synthesis, reliability, facility of fault detection etc., which makes their study significant from a practical point of view.

0-7695-0161-3/99 $10.00 © 1999 IEEE

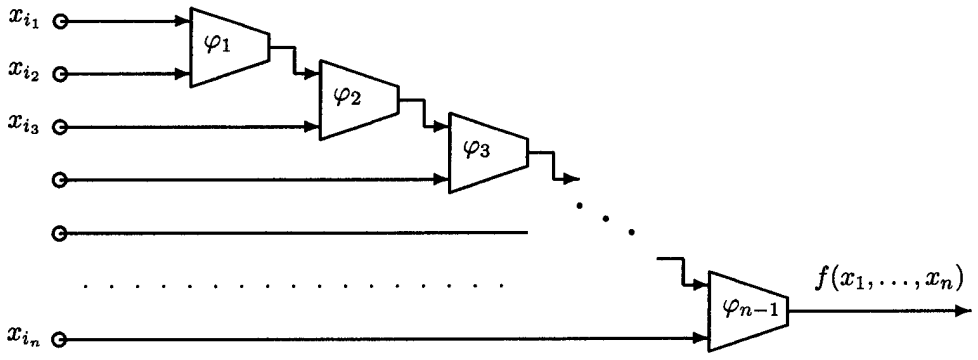


Figure 1. Cascade circuit

Among "most decomposable" functions, those which can be formally expressed through the least possible number of basic operations are naturally included. Considering n-variable essential functions and $\Omega(2)$ as the domain of operations this minimal number of gates is $n-1$. Such functions, i.e., those realizable by n-input $(n-1)$-gate combinational circuits are called *ortholinear* [17], and such circuits are called *disjunctive networks* [12].

The focus of our study here is a subclass of ortholinear functions, which can be realized by so called "cascades" (see Figure 1).

Definition 1 *Function $f(x_1,\ldots,x_n) \in \Omega^e(n)$ is called n-cascade, or simply cascade if there is a permutation $\pi = (i_1,\ldots,i_n)$ and $n-1$ operations $\varphi_1,\ldots,\varphi_{n-1} \in \Omega(2)$, such that*

$$f(x_1,\ldots,x_n) = \varphi_{n-1}(x_{i_n}, \varphi_{n-2}(x_{i_{n-1}}, \varphi_{n-3}\cdots \ldots\varphi_1(x_{i_2}, x_{i_1}))\ldots)).$$

Let $\Upsilon(n)$ be the set of all cascade functions in $\Omega^e(n)$. Put $C(n) := |\Upsilon(n)|$ – the number of all n-cascades.

The properties of cascades have been studied for many years and by many researchers ([1] – [13]). Several papers (e.g. [3, 6, 9, 11, 12, 13]) have focused on the number of all cascades and/or their special types. Particularly, J.T.Butler [12] established a recurrence relation

$$C(n) = \sum_{i=1}^{n-1} (-1)^{i+1} \binom{n}{i} (2^{i+1}+1) C(n-i) - (-1)^n (2^n+1) C(1), \qquad (2)$$

as well as an asymptotic approximation –

$$C(n) \simeq n! a^n b \qquad \text{for large } n,$$

where $a = 4.04095\cdots$, $b = 0.28790\cdots$.

Expression (2) renders a simple algorithm for computation of the number $C(n)$. This algorithm was implemented in a computer program and the list of first 15 numbers for $C(n)$ appeared in [12].

In spite of the substantial progress so far in understanding and enumerating cascade functions there has been no explicit formula discovered for the number $C(n)$. The main goal of this paper is to prove such a formula, and thus to explicitly solve the enumeration problem for this class of logic functions.

Before proceeding to the main result presented in the next section, a few more definitions are needed.

Definition 2 *Function $f(x_1,\ldots,x_n) \in \Omega^e(n)$ is called (n,k)-cascade, $1 \le k \le n$, if there exists a subset of the variable set $X' = \{x_{i_1},\ldots,x_{i_k}\} \subseteq X = \{x_1,\ldots,x_n\}$ and a subfunction $g(X \setminus X')$, such that*

$$f(x_1,\ldots,x_n) = \varphi_k(x_{i_k}, \varphi_{k-1}(x_{i_{k-1}}, \ldots \varphi_1(x_{i_1}, g)\ldots)).$$

for some operations $\varphi_1,\ldots,\varphi_k \in \Omega(2)$.

Note, that the position of g is significant, i.e., not every $k+1$-cascade function $h(g, x_{i_1},\ldots,x_{i_k})$ is an (n,k)-cascade.

Note also, that for the sake of formal consistency of previous definitions a cascade function could have been defined in a more general way, to be

$$f(x_1,\ldots,x_n) = \varphi_n(x_{i_n}, \varphi_{n-1}(\ldots \varphi_2(x_{i_2}, \varphi_1(x_{i_1})\ldots)),$$

for a monadic operation $\varphi_1 \in \Omega(1)$ and some $n-1$ dyadic operations $\varphi_2,\ldots,\varphi_n \in \Omega(2)$. It is easy to see though that the two definitions produce the same class of functions, as the existence of $\varphi_2(x_{i_2}, \varphi_1(x_{i_1}))$ induces the existence of $\phi(x_{i_2}, x_{i_1})$ and vice versa.

Clearly, an (n,n)-cascade is simply a cascade function. Moreover, since all one- and two-variable essential functions are cascades, any $(n,n-1)$- or $(n,n-2)$-cascade is also a cascade.

Another extreme case, when $k=1$, defines an important type of functions – $(n,1)$-cascades. These functions can be expressed as follows

$$f(x_1,\ldots,x_n) = g(x_{i_1},\ldots,x_{i_{n-1}}) \circ x_{i_n},$$

for a dyadic operation $\circ$.

By $C_1(n)$ we denote the number of all $(n,1)$-cascades.

Definition 3 *For any function $g(X) \in \Omega(n)$ and any variable set $Y = \{y_1,\ldots,y_m\} : Y \cap X = \emptyset$ a $(n+m,m)$-cascade f –*

$$f(x_1,\ldots,x_n) = h(g,y_1,\ldots,y_m), \qquad h \in \Upsilon(m+1)$$

is called an m-cascade extension of g.

2. The Number of Cascade Functions

Theorem 1 *The number $C(n)$ of n-variable cascade functions $n \geq 2$ holds*

$$C(n) = C_1(n) - \sum_{m=1}^{n-1} (E(n-m) - C_1(n-m)) \binom{n}{m} \hat{C}(m), \quad (3)$$

where $E(n)$ is the number of essential functions given by (1);

$$C_1(n) = \sum_{i=1}^{n-1} (-1)^{i+1} \binom{n}{i} (2^{i+1}+1) E(n-i) - (-1)^n (2^{n+1}+2); \quad (4)$$

$$\hat{C}(k) = k! \sum_{1r_1+\cdots+kr_k=k} (-1)^{k-(r_1+\cdots+r_k)} \times (r_1+\cdots+r_k)! \prod_{i=1}^{k} \frac{(2^{i+1}+1)^{r_i}}{r_i!(i!)^{r_i}}. \quad (5)$$

Proof

A. The Formula. First the general approach of counting all cascade functions is established. The structure of the formula (3), clearly shows the method. The number of n-cascades is equal to the number of all $(n,1)$ cascades $C_1(n)$ minus the sum (by $m = 1\ldots n-1$) of the number of all functions such that each one is an (n,m)-cascade, but the corresponding subfunction is not an $(n-m,1)$-cascade. The latter sum is expressed by

$$\sum_{m=1}^{n-1} (E(n-m) - C_1(n-m)) \binom{n}{m} \hat{C}(m),$$

where $\hat{C}(n)$ is the number of m-cascade extensions for a chosen non-$(n-m,1)$-cascade subfunction. This number $\hat{C}(n)$ is multiplied by the number of choices $\binom{n}{m}$, and by the number of subfunctions $E(n-m) - C_1(n-m)$.

B. $C_1(n)$. This method has been chosen particularly because of the simplicity of counting the number $C_1(n)$. The formula (4) is built using the principle of inclusion and exclusion. First we take the number of all 1-cascade extensions of all essential $n-1$-variable functions. This number is $5\binom{n}{1}E(n-1)$. It is easy to see that for each chosen variable there are only 5 ways of 1-cascade extensions, considering all 10 essential dyadic operations. Then the principle is applied by subtracting and adding the number of all i-cascade extensions of $(n-i)$-variable essential functions, counting only those symmetrically extendable regarding all i variables.

$E(n-i)$ is multiplied by the number $\binom{n}{i}$ of choices of i variables, as well as by the number $(2^{i+1}+1)$ of all i-cascade symmetric extensions, i.e.,

$$f(x_1,\ldots,x_n) = g(x_1,\ldots,x_{n-i}) \circ \Phi(x_{n-i+1},\ldots,x_n),$$

where $\Phi(x_{n-i+1},\ldots,x_n)$ is a totally symmetric cascade. There are 2^i different ways to choose an i-tuple $(a_{n-i+1},\ldots,a_n)$ so that

$$f(x_1,\ldots,x_{n-i},a_{n-i+1},\ldots,a_n) \equiv g(x_1,\ldots,x_{n-i}).$$

If $\circ$ is a so-called "strong operation", i.e., one of the following eight – $\wedge$=AND, $\vee$=OR, $\neg\wedge$=NAND, $\neg\vee$=NOR, $>, <, \geq, \leq$, then for all other values of $(x_{n-i+1},\ldots,x_n)$ the function f is a constant 0 or 1. This makes the number of choices at least $2 \cdot 2^i = 2^{i+1}$. Finally, there is only one (essential) extension when $\circ$ is $\oplus$=XOR or $\sim$. Thus the number of all possibilities is $2^{i+1}+1$.

Similar constructions as above are used in [12] for establishing the recurrent expression (2). This explains the resemblance of our formula for $C_1(n)$ with (2).

C. $\hat{C}(k)$. This is the core part of the formula, since by counting all k-cascade extensions we deal with the essence of cascade structures.

The formula (5) is obtained as result of a simplifica-

n	$E(n)$	$C_1(n)$	$\hat{C}(n)$	$C(n)$
1	2	2	5	2
2	10	10	41	10
3	218	114	497	114
4	64,594	3,922	8033	1,842
5	4,294,642,034	1,596,666	162,305	27,226
6	$\approx$1.85e20	128,830,610,650	3,935,201	902,570
7	$\approx$3.4e39	$\approx$6.46e21	111,313,697	25,530,658
8	$\approx$1.158e78	$\approx$1.36e41	-	825,345,250
9	$\approx$1.34e117	$\approx$5.21e79	-	30,016,622,298
10	$\approx$1.8e311	$\approx$6.7e156	-	1,212,957,186,330

Table 1. Number of n-variable cascade functions.

tion of the following expression

$$\hat{C}(k) = \sum_{t=1}^{k} (-1)^{t-1}(k-t+1)! \times \sum_{a_1 r_1 + \cdots + a_t r_t = k} (2^{a_1+1}+1)^{r_1} \cdot \ldots \cdot (2^{a_t+1}+1)^{r_t} \times \frac{k!}{r_1!(a_1!)^{r_1} \cdot \ldots \cdot r_t!(a_t!)^{r_t}}.$$

The first sum indicates that we apply again the inclusion and exclusion method. The first member of the sum is $5^k \cdot k!$, which is the number of all orderings of k variables, times the number of k-cascade extensions with respect to each order. This number is just an upper bound for $\hat{C}(k)$, i.e., $\hat{C}(k) \leq 5^k \cdot k!$, since there are functions that are counted more than once. Note that, this simple upper bound is quite close the asymptotic formula found in [12] and cited in the previous section.

We then subtract and add k-cascade extensions which have 2 (minus), 3(plus), ... symmetric variables. The second sum is taken for all t-block partitions of the number k. These partitions correspond to the partitions of the k variable set induced by the symmetry relations. The fractional expression is the number of partitions which themselves add up to the Bell number. They are multiplied by the number of extensions with corresponding t-block symmetries. The expression $(2^{a_i+1}+1)^{r_i}$ represents r_i copies of the same a_i-block, each of which is counted by using the same argument as the one used in part B of this proof. □

3. Concluding Remarks

For handling large numbers a mathematical tool such as Maple or Mathematica can now help to calculate the number of cascade functions depending on any feasible number variables. The data of Table 1 (computed with Maple V) shows the numerical values for $C(n)$ and for the coefficients involved, for $n = 1, 2, \ldots, 10$.

The numbers $E(n)$ for $n \geq 6$ and $C_1(n)$ for $n \geq 7$ are very large (≥ 20 digits), and are thus presented in the table as approximations. The coefficient $\hat{C}(n)$ has been computed for up to $\hat{C}(7)$, which is sufficient for counting $C(10)$.

It is important to note that the numbers for $C(n)$ above are exactly the same as those obtained by using a computer program implementing the recurrence expression in [12].

After certain modification the method for counting cascades shown in this paper can be used for establishing further enumeration results for some classes of logic functions, such as ortholinear functions (disjunctive networks), multiple-valued cascades and multiple-valued ortholinear functions. We would like to note also, that after substitution of all its formal subexpressions, the main formula for $C(n)$ can be simplified and presented in a compact form of n.

References

[1] Maitra, K.K., "Cascade switching networks of two-input flexible cells," *IRE Trans. Electron. Comp.*, Vol. EC-11, pp.136-143, 1962.

[2] Sklansky, J., "General synthesis of tributary switching networks," *IEEE Trans. Electron. Comput.*, Vol. EC-12, pp.464-469, 1963.

[3] Sklansky, J., Korenjak, A.J., Stone, H.S., "Canonical tributary networks," *IEEE Trans. Electron. Comput.*, Vol. EC-14, pp.961-963, 1965.

[4] Stone, H.S., Korenjak, A.J., "Canonical form and synthesis of cellular cascades," *IEEE Trans. Electron. Comput.*, Vol. EC-14, pp.852-862, 1965.

[5] Minnick, R.C., "Cutpoint cellular logic," *IEEE Trans. Electron. Comput.*, Vol. EC-13, pp.685-698, 1964.

[6] Stone, H.S., "On the number of equivalent classes of functions realizable by cellular cascades," *Proceedings Nat. Symp. Impact of Batch Fabrication on Future Computers*, IEEE Publ. 1601, pp.81-87, 1965.

[7] Weiss, C.D., "The characterization and properties of cascade realizable switching functions," *IEEE Trans. Comput.*, Vol. C-18, pp.625-633, 1969.

[8] Mukhopadhyay, A., "Unate cellular logic," *IEEE Trans. Comput.*, Vol. C-18, pp.625-633, 1969.

[9] Frécon, L., "Capacité logique des structures en ligne et en peigne d'operatures booleens a deux variables," *Revue Francaise d'Informatique et de Recherche Operationnelle*, No.B-2, pp.62-85, 1971.

[10] Papakonstantinou, G.K., "A synthesis method for cutpoint networks," *IEEE Trans. Comput.*, Vol. C-21, pp.1286-1292, 1972.

[11] Chakrabarti K., Kolp, O., "Fan-in constrained tree networks of flexible cells," *IEEE Trans. Comput.*, Vol. C-23, pp.1238-1249, 1972.

[12] Butler, J.T., "On the number of functions realized by cascades and disjunctive networks," *IEEE Trans. Comput.*, Vol. C-24, No.7, pp.681-690, 1975.

[13] Sasao, T., Kinoshita, K., "On the number of fanout-free functions and unate cascade functions," *IEEE Trans. Comput.*, Vol. C-28, No.1, pp.866-872, 1979.

[14] Muroga, S., "Logic Design and Switching Theory," *John Wiley & Sons*, 1979.

[15] Sasao, T. (ed.), "Logic Synthesis and Optimization," *Kluwer Academic Publishers*, 1993.

[16] Sasao, T. and Fujita, M. (eds.), "Representations of Discrete Functions," *Kluwer Academic Publishers*, 1996.

[17] Pogosyan, G., "On ortholinear functions," *Note on Multiple-Valued Logic in Japan*, Vol. 19, No.4, pp.1-10, 1996.

Research on the Similarity among Precomplete Sets Preserving m-ary Relations in Partial K-Valued Logic*

Renren Liu
Dept. of Computer Science
Xiangtan University
Hunan, 411105, China
E-mail: Liurr@public.xt.hn.cn

Abstract

In multiple-valued logic theories, the characterization of Sheffer[1] functions is an important problem, it includes the decision and construction for Sheffer functions in P_k and P_k^*. The solution of these problems depends on the solution of the decision problem for completeness in P_k and P_k^*, and reduced to determining the minimal coverings of precomplete sets in P_k and P_k^*, respectively. this paper studies some similarity among precomplete sets preserving m-ary relation in P_k^*.

Keywords: Partial multiple-valued logic, Completeness, Sheffer functions.

I. Introduction

Let $E_k = \{0,1,\cdots,k-1\}, k \geq 2$. The total and partial k-valued logic functions over E_k are called partial k-valued logic functions. The set of all partial k-valued logic functions is denoted by P_k^*, namely

$$P_k^* = \{f(x_1,\cdots,x_n) \mid f: E_k^n \to E_k \cup \{*\}, \quad n \geq 1\}$$

where $f(\alpha_1,\cdots,\alpha_n) = *$ means that f is undefined at point $(\alpha_1,\cdots,\alpha_n) \in E_k^n$. We denoted by * the empty function that is undefined at every point.

A Function $f(x_1,\cdots,x_n) \in P_k^*$ is said to be a Sheffer function if any function in P_k^* can be composed by $f(x_1,\cdots,x_n)$.

*This work is supported by a grant from the Natural Science Fund of Hunan Province ,China A function

It follows from the theory on completeness [2] that f is a Sheffer function if, and only if, for any precomplete set $A \subset P_k^*$ there have $f \notin A$. But the number of precomplete sets in P_k^* will raise rapidly with increment of k, so it is necessary to find a simpler method for decision on Sheffer functions.

Although any two precomplete sets A and B in P_k^* hold that $A \not\subset B$ and $B \not\subset A$, one precomplete set may be included in union of some precomplete sets. Thus when we decide whether f is a Sheffer function , those precomplete sets which are included in the union of some precomplete sets need not to be checked whether f is included in them. Therefore the decision on Sheffer functions is reduced to determining the minimal coverings of precomplete sets.

The problem of determining the minimal covering of precomplete sets in P_3^* was solved in 1991 by the author [3], and for $k \geq 4$, the author has proved that T_E, $L_{G_{4,2}}$ *and* L_p are the component parts of the minimal covering of precomplete sets in P_k^* ([4], [5], [6]).

In this paper, we shall study the similarity among precomplete sets preserving m-ary relation in partial k-valued logic.

II. Some Definitions

Let

$$f(t_1,\cdots,t_m), g_i(x_1,\cdots,x_n) \in P_k^*, i = 1,\cdots,m.$$

The composition

$$h(x_1,\cdots,x_n) = f(g_1(x_1,\cdots,x_n),\cdots,g_m(x_1,\cdots,x_n))$$

is defined as follows:

For arbitrary $\overline{\alpha} = (\alpha_1,\cdots,\alpha_n) \in E_k^n$,

0-7695-0161-3/99 $10.00 © 1999 IEEE

if $g_1(\overline{\alpha}),\cdots,g_m(\overline{\alpha})$ are all defined, then $h(\overline{\alpha})=f(g_1(\overline{\alpha}),\cdots,g_m(\overline{\alpha}))$; and if one of $g_1(\overline{\alpha}),\cdots,g_m(\overline{\alpha})$ is not defined, then $h(\overline{\alpha})=*$ (undefined).

Let $f(t_1,\cdots,t_m)\in A\subseteq P_k^*$, The function $f(g_1(x_1,\cdots,x_n),\cdots,g_m(x_1,\cdots,x_n))$ is said to be composed from A if $g_i(x_1,\cdots,x_n)(i=1,\cdots,m)$ either belongs to A or is one of the variables $x_1,\cdots x_n$.. The set of all functions composed from A is denoted by A'.

$A(\subset P_k^*)$ is said to be closed if $A=A'$.

A is said to be complete if $\overline{A}=P_k^*$, where $\overline{A}=A'\cup(A')'\cup((A')')'\cup\cdots$.

Let $M(\subset P_k^*)$ be a closed set, if there is no other closed set between M and P_k^*, then M is said to be a precomplete set or maximal closed set.

Let Σ be the set of all precomplete sets in P_k^*, $\Sigma'\subset\Sigma$. We denote by $S_{\Sigma'}$ the set of all functions $f\in A_j$, where $A_j\in\Sigma'$, namely,

$$S_{\Sigma'}=\{f\in P_k^*\,|\,\exists A_j((A_j\in\Sigma')\wedge(f\in A_j))\}$$

Set $\Sigma'(\subset\Sigma)$ is said to be the covering of Σ if $S_{\Sigma'}=S_\Sigma$.

Covering Σ' is said to be minimal if any proper subset of Σ' is not the covering of Σ.

Obviously, for a precomplete set $A\subset P_k^*$, if there exists a function $f\in A$ such that for any other precomplete set $B(\subset P_k^*)$, $f\notin B$ then A must appear in the minimal covering of precomplete sets of P_k^*, so we call A the component part of the minimal covering of precomplete sets of P_k^*.

Let $\varnothing\subset E\subset E_k$. The function $f(x_1,\cdots,x_n)$ is said to preserve E if $f(a_1,\cdots,a_n)=*$ or $f(a_1,\cdots,a_n)^*$ belongs to E for arbitrary $a_1,\cdots,a_n\in E$. The set of all functions preserving E is denoted by T_E.

A set of some m-tuples $<a_1,\cdots,a_m>$, $a_i\in E_k$ is called an m-ary relation over E_k, denoted by G_m. $\overline{a}^1=(a_1^1,\cdots,a_n^1),\cdots,\overline{a}^m=(a_1^m,\cdots,a_n^m)$ are said to preserve the m-ary relation G_m if $<a_i^1,\cdots,a_i^m>\in G_m$, $i=1,\cdots,n$; a function $f(x_1,\cdots,x_n)$ is said to preserve the m-ary relation G_m if for arbitrary $\overline{a}^1,\cdots,\overline{a}^m$ preserving G_m either $<f(\overline{a}^1),\cdots,f(\overline{a}^m)>\in G_m$ or one of $f(\overline{a}^1),\cdots,f(\overline{a}^m)$ is undefined. The set of all functions preserving G_m is denoted by $T(G_m)$. Hereafter we always assume $2\le m\le k$.

For the sake of unity, we look on T_E as a set of functions preserving m-ary relation G_m. Suppose without loss of generality that $E=\{0,1,\cdots,s-1\}$, $1\le s\le k-1$, define m-ary relation as follows:

$$G_m=\{<i,a_1,\cdots,a_{m-1}>\,|\,i=0,1,\cdots,s-1,a_j\in E_k,j=1,\cdots,m-1\}$$

Obviously, $|G_m|=s\times k^{m-1}\le(k-1)k^{m-1}<k^m$ and there exist $<a_1,\cdots,a_m>\in G_m$ such that $a_i\ne a_j,i\ne j,i,j=1,\cdots,m$, therefore G_m is a non-total relation, i.e., $T(G_m)<P_k^*$. We shall prove $T_E=T(G_m)$ as follows:

Take $f(x_1,\cdots,x_n)\in T_E$, thereby for $a_1,\cdots,a_n\in E$, $f(a_1,\cdots,a_n)=a,a\in E$ or $f(a_1,\cdots,a_n)=*$. For any $\overline{a}^1=(a_1^1,\cdots,a_n^1),\cdots,\overline{a}^m=(a_1^m,\cdots,a_n^m)$ which preserves G_m, since $<a_i^1,\cdots,a_i^m>\in G_m,$, $a_i^1\in E,i=1,\cdots,n$

Thus $f(\overline{a}^1)=f(a_1^1,\cdots,a_n^1)\in E\cup\{*\}$ and $f(\overline{a}^i)=f(a_1^i,\cdots,a_n^i)\in E_k\cup\{*\},i=2,\cdots,m$. It follows from definition of the G_m that $<f(\overline{a}^1),\cdots,f(\overline{a}^m)>\in G_m$ or one of $f(\overline{a}^1),\cdots,f(\overline{a}^m)$ is undefined. Therefore $f\in G_m$. Consequently, $T_E\subseteq T(G_m)$. Since that T_E is the precomplete set and $T(G_m)<P_k^*$, $T_E=T(G_m)$.

Relation G_m and G_m' is said to be similar if $|G_m|=|G_m'|$ and there exist a one-one mapping $\sigma:E_k\to E_k$ such that

$$G_m'=\{<\sigma(a_1),\cdots,\sigma(a_m)>\,|\,<a_1,\cdots,a_m>\in G_m\}$$

Clearly, similar relation defined above is equivalent relation. We denote by $G_m\cong_\sigma G_m'$ that G_m is similar to G_m' under one-one mapping σ.

III. Main Results and Proof

THEOREM 1. Let $T(G_m)$ be a precomplete set except $P_k^* \cup \{*\}$ and $G_m \cong_\sigma G'_m$. Then $T(G'_m)$ is also a precomplete set.

Proof. From [2], all precomplete sets except $P_k^* \cup \{*\}$ in P_k^* are classified to 6 classes, they are as follows:

$T_E, F_{S,m}, S_{I,m}, S_{R,m}, L_{G_{4,2}}(V_{G_{4,2}})$ and L_p.

1. Let $T(G_m) = T_E$. Suppose without loss of generality that $E = \{0,1,\cdots,s-1\}, 1 \le s \le k-1$, then

$$G_m = \{< i, a_1, \cdots, a_{m-1} > \mid i = 0,1,\cdots,s-1, a_j \in E_k, j = 1,\cdots,m-1\}$$

It follows from the definition of similar relation that

$$G'_m = \{< \sigma(i), \sigma(a_1), \cdots, \sigma(a_{m-1}) > \mid < i, a_1, \cdots, a_{m-1} > \in G_m\}$$

Let $E' = \{\sigma(0), \cdots, \sigma(s-1)\}$, it is easy to prove that $T(G'_m) = T_{E'}$. Therefore $T(G'_m)$ is a precomplete set.

2. Let $T(G_m) = F_{S,m}$. Then

$G_m = \bigcup_{\substack{i,j=1 \\ i \ne j}}^{m} G_m(\{i,j\}) \cup G_m^*$, where G_m^* is either empty ($m \ne 2$) or only contains such m-ary sequences $< a_1, \cdots, a_m >$ that $a_i \ne a_j, i \ne j, i, j = 1, \cdots, m$, and G_m is symmetric to S_m. We have by the definition of similar relation that

$$G'_m = \{< \sigma(a_1), \cdots, \sigma(a_m) > \mid < a_1, \cdots, a_m > \in G_m\}$$

Since σ is one-one, we have

$$G'_m = \bigcup_{\substack{i,j=1 \\ i \ne j}}^{m} G'_m(\{i,j\}) \cup G_m^{*\prime}$$

where

$$G'_m(\{i,j\}) = \{< \sigma(a_1), \cdots, \sigma(a_m) > \mid < a_1, \cdots, a_m > \in G_m, \sigma(a_i) = \sigma(a_j)\}$$

$, i \ne j, 1 \le i, j \le m,$

$$G_m^{*\prime} = \{< \sigma(a_1), \cdots, \sigma(a_m) > \mid < a_1, \cdots, a_m > \in G_m^*\}$$

We prove G'_m is symmetric to S_m as follows:

Suppose $\tau \in S_m, < b_{\tau(1)}, \cdots, b_{\tau(m)} > \in G_m'^{\tau}$, i.e., $< b_1, \cdots, b_m > \in G'_m$. By definition we have that $< \sigma^{-1}(b_1), \cdots, \sigma^{-1}(b_m) > \in G_m$. Let $c_i = \sigma^{-1}(b_i), \ i = 1, \cdots, m$, then $< c_1, \cdots, c_m > \in G_m$. Since $G_m^{\tau} = G_m$, $< c_{\tau(1)}, \cdots, c_{\tau(m)} > \in G_m$. Therefore $< \sigma(c_{\tau(1)}), \cdots, \sigma(c_{\tau(m)}) > \in G'_m$. Obviously, $c_{\tau(i)} = \sigma^{-1}(b_{\tau(i)}), \ i = 1, \cdots, m$. So

$$\sigma(c_{\tau(i)}) = \sigma(\sigma^{-1}(b_{\tau(i)})) = b_{\tau(i)}, \quad i = 1, \cdots, m.$$

Thus,

$$< b_{\tau(1)}, \cdots, b_{\tau(m)} > = < \sigma(c_{\tau(1)}), \cdots, \sigma(c_{\tau(m)}) > \in G_m'$$

Consequently, $G_m'^{\tau} \subseteq G'_m$.

On the other hand, suppose $< b_1, \cdots, b_m > \in G'_m$, then $< \sigma^{-1}(b_1), \cdots, \sigma^{-1}(b_m) > \in G_m$. Since $G_m^{\tau} = G_m$, $G_m^{\tau^{-1}} = G_m$. Thereby

$$< \sigma^{-1}(b_{\tau^{-1}(1)}), \cdots, \sigma^{-1}(b_{\tau^{-1}(m)}) > \in G_m$$

$$\Rightarrow < b_{\tau^{-1}(1)}, \cdots, b_{\tau^{-1}(m)} > \in G_m'$$

$$\Rightarrow < b_1, \cdots, b_m > \in G_m'^{\tau}$$

$$\Rightarrow G'_m \subseteq G_m'^{\tau}$$

It means $G_m'^{\tau} = G'_m$, i.e., G'_m is symmetric to S_m.

3) For the case that $T(G_m)$ is one of $S_{I,m}, S_{R,m}, L_{G_{4,2}}(V_{G_{4,2}})$ and L_p, the proof is similar to the proof given above.

THEOREM 2. Suppose $G_m \cong_\sigma G'_m$ and $f(x_1, \cdots, x_n) \in T(G_m)$. Let

$$g(x_1, \cdots, x_n) = \sigma(f(\sigma^{-1}(x_1), \cdots, \sigma^{-1}(x_n))), \sigma(*) = *$$

Then $g(x_1, \cdots, x_n) \in T(G'_m)$.

Proof. Suppose that $\bar{a}^1, \cdots, \bar{a}^m$ preserve G'_m, where $\bar{a}^i = (a_1^i, \cdots, a_n^i), i = 1, \cdots, m$. Thus $< a_i^1, \cdots, a_j^m > \in G'_m, j = 1, \cdots, n$. From definition we have that

$$g(a_1^i, \cdots, a_n^i) = \sigma(f(\sigma^{-1}(a_1^i), \cdots, \sigma^{-1}(a_n^i))), \ i = 1, \cdots, m$$

and $< \sigma^{-1}(a_j^1), \cdots, \sigma^{-1}(a_j^m) > G_m, j = 1, \cdots, n$.

Thereby.

$$< f(\sigma^{-1}(a_1^1), \cdots, \sigma^{-1}(a_n^1)), \cdots, f(\sigma^{-1}(a_1^m), \cdots, \sigma^{-1}(a_n^m)) > \in G_m$$

So

$$< g(\bar{a}^1), \cdots, g(\bar{a}^m) >$$
$$= < \sigma(f(\sigma^{-1}(a_1^1), \cdots, \sigma^{-1}(a_n^1))), \cdots, \sigma(f(\sigma^{-1}(a_1^m), \cdots, \sigma^{-1}(a_n^m))) > \in G'_m$$

Therefore $g(x_1, \cdots, x_n) \in T(G'_m)$.

THEOREM 3. Suppose that $G_m \cong_\sigma G'_m$ and $T(G_m) \subseteq \bigcup_{i=1}^{s} T(G''_{h_i})$ (Union of some precomplete sets), $2 \le h_i \le k$, and when $h_i = m$, $T(G_m) \neq T(G''_m)$. Then $T(G'_m) \subseteq \bigcup_{i=1}^{s} T(G'''_{h_i})$, where $G''_{h_i} \cong_\sigma G'''_{h_i}$, and when $h_i = m$, $T(G'_m) \neq T(G'''_m)$.

Proof. Suppose $g(x_1,\cdots,x_n) \in T(G'_m)$. Let. $f(x_1,\cdots,x_n) = \sigma^{-1}(g(\sigma(x_1),\cdots,\sigma(x_n))),\ \sigma(*) = *$ Then following after the proof of theorem 2, we can prove that $f(x_1,\cdots,x_n) \in T(G_m) \subseteq \bigcup_{i=1}^{s} T(G''_{h_i})$.

Without loss of generality suppose $f \in T(G'_{h_i})$. Let $G''_{h_i} \cong_\sigma G'''_{h_i}$, from the definition of f, we have $g(x_1,\cdots,x_n) = \sigma(f(\sigma^{-1}(x_1),\cdots,(\sigma^{-1}(x_n))),\ \sigma(*) = *$ Thus it follows from theorem 1 and theorem 2 that $T(G'''_{h_i})$ is a precomplete set and $g(x_1,\cdots,x_n) \in T(G'''_{h_i})$.

When $h_i = m$, we know that $T(G_m) \neq T(G''_m)$. If $T(G'_m) = T(G'''_m)$.

Suppose $f(x_1,\cdots,x_n) \in T(G_m)$. Let $g(x_1,\cdots,x_n) = \sigma(f(\sigma^{-1}(x_1),\cdots,(\sigma^{-1}(x_n))),\ \sigma(*) = *$ Then it follows from theorem 2 that $g(x_1,\cdots,x_n) \in T(G'_m) \subseteq T(G'''_m)$, *i.e.*, $g(x_1,\cdots,x_n) \in T(G'''_m)$.

But, $f(x_1,\cdots,x_n) = \sigma^{-1}(g(\sigma(x_1),\cdots,\sigma(x_n))),\ \sigma(*) = *$ since $G''_m \cong_\sigma G'''_m$, following after the proof of theorem 2 we have $f(x_1,\cdots,x_n) \in T(G''_m)$, so $T(G_m) \subseteq T(G''_m)$. It can be proved in the same way that $T(G''_m) \subseteq T(G_m)$.

Therefore $T(G_m) = T(G''_m)$, which is a contradiction. Hence $T(G'_m) \neq T(G'''_m)$.

Following after the proof of theorem 2 and theorem 3, we have

THEOREM 4. Let $G_m \cong_\sigma G'_m$ and there exist $f(x_1,\cdots,x_l) \in T(G_m)$ such that for arbitrary $T(I_n) \neq T(G_m)$ there have $f \notin T(I_n)$. Then there exist $g(x_1,\cdots,x_l) \in T(G'_m)$ such that for arbitrary $T(I'_n) \neq T(G'_m)$ there have $g \notin T(I'_n)$, where I_n, I'_n are the set of n-ary relations, $2 \le n \le k$.

The four theorems given above show that two sets of functions preserving two similar relations respectively are either all the component parts of the minimal covering of precomplete sets of P_k^* or all not.

References

[1] Sheffer, H.M.(1913). "A Set of Five Independent Postulates for Boolean Algebras with Application to Logical Constants", *Trans. Am. Math. Soc.*, 14, pp. 481-488.

[2] Czukai Lo (Zhukai Luo), (1984). "The Completeness Theory of Partial Many-Valued Logic Functions", *Acta Math.* Sinica., 27, 5, 676-683; MR 87a:03049a 03B50.

[3] Renren Liu,(1991). "The Minimal Covering of Precomplete Sets in Partial Three-Valued Logic", *Natural Sciences J.* Xiangtan Univ., 13(2), 158-165; MR92g:03035 03B50.

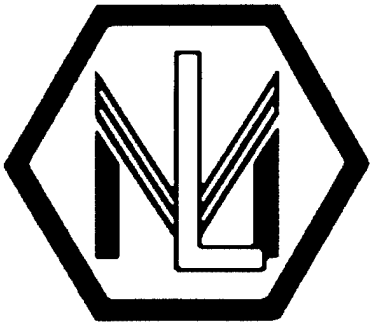
[4] Renren Liu, (1992). "Some Results on the Minimal Covering of Precomplete Sets in Partial K-Valued Logic(I)", *Natural Sciences J.* Xiangtan Univ.,14(1),123-129; MR93h:03023 03B50.

[5] Renren Liu, (1993). "Some Results on the Minimal Covering of Precomplete Sets in Partial K-Valued Logic(II)", *Natural Sciences J.* Xiangtan Univ.,15(2), pp. 141-147.

[6] Renren Liu, (1996). "Some Results on the Decision for Sheffer Functions in Partial K-Valued Logic", *Multiple Valued Logic-An International Journal*, Vol.1, pp. 253-269.

[7] Renren Liu, (1998). "Some Results on the Decision for Sheffer Functions in Partial K-Valued Logic", *Proceedings of the 28th International Symposium on Multiple-Valued Logic*, pp. 77-81.

SESSION V
INVITED ADDRESS
CHAIR: Elena Dubrova



Structural and Behavioral Modeling with Monadic Logics

Abdelwaheb Ayari, David Basin and Stefan Friedrich
Institut für Informatik, Albert-Ludwigs-Universität Freiburg,
Universitätsgelände Flugplatz, 79110 Freiburg i. Br., Germany.

Abstract

Logic offers the possibility of modeling and reasoning about hardware and software. But which logic? We propose monadic logics of strings and trees as good candidates for many kinds of discrete systems. These logics are natural, decidable, yet substantially more expressive, extensions of Boolean logic. We motivate their applicability through examples.

1 Introduction

Transistors, pins, and power supplies are the stuff from which computers are made. However, this view is not very helpful for understanding what computers actually do. For this we require a language for modeling the structure and behavior of systems and for relating these models, i.e., for showing that the system modeled has the intended behavior. This requirement is fulfilled by logic.

Boolean logic is the traditional logic of switching theory. It is often applied to model and reason about discrete systems. From the engineering perspective, it is a logic for expressing operations on bits. For instance, a combinational circuit may be represented in Boolean logic by a formula whose syntax reflects its structure, e.g., an *and*-gate in the circuit corresponds to a logical and-operation. Moreover, one can model the circuit's behavior in the logic and verify that the two models are equivalent. When the models are accurate, the result predicts the behavior of actual circuits.

The primacy of Boolean logic for such tasks is understandable. It is conceptually simple and adequate for modeling the structure and behavior of many kinds of discrete systems. Moreover, verification is decidable and, using data-structures like BDDs, many problems have acceptable running times. However there are drawbacks. Certain kinds of systems, even when discrete (or viewed as such) cannot be modeled in Boolean logic. Moreover, even when they can, the Boolean model may differ substantially from an engineer's model, and this makes modeling error prone. A solution is to use more expressive logics like first-order logic, trading increased expressiveness for the loss of decidability. Here we propose a middle way, namely, decidable monadic second-order logics of strings and trees [5, 8, 14, 17].

We motivate this proposal by examining the potential and limitations of structural and behavioral modeling in a progression of logics, from Boolean logic to extensions with quantifiers and variables ranging over strings and trees of Booleans. The string and tree logics may seem exotic, but they are natural and significant extensions of Boolean logic and have a much wider range of applications. Moreover, like Boolean logic, they are decidable and hence offer the possibility of automated verification. Thus, although monadic logics may not challenge the primacy of Boolean logic, we believe that they are well-suited for reasoning about discrete systems and deserve a place in the repertoire of engineers working with such systems.

Although the use of the monadic logic of strings in modeling parameterized combinational and sequential systems has already been established (see [3, 4]), the exposition here is from a different perspective, centered around exploring tradeoffs and providing support for the above proposal. In addition, the use of a monadic logic of trees for hardware modeling and verification is new and presents a new approach to reasoning about hierarchical, tree-structured systems.

2 Boolean Logic

We begin with Boolean logic and its application to structural and behavioral modeling. Although this logic is adequate for many problems, it has limitations, which motivate generalizations.

The *formulae* of Boolean logic are built from *variables* from a set $\mathcal{V} \equiv \{x_1, x_2, \dots\}$ and *functions* (also called *connectives*) like $\vee$, $\wedge$, $\rightarrow$, $\leftrightarrow$, and $\neg$. Formulae are interpreted in the set $\mathbb{B} \equiv \{0, 1\}$ of *truth values*.

0-7695-0161-3/99 $10.00 © 1999 IEEE

An *evaluation* $\sigma : \mathcal{V} \to \mathbb{B}$ is a mapping from variables to truth values that is homomorphically extended to formulae. When $\sigma(\varphi) = 1$, for some σ, we say that φ is *true under* σ and that φ is *satisfiable*. Since the satisfiability of $\varphi(x_1, \ldots, x_n)$[1] depends only on the assignments for the variables $x_1, \ldots, x_n$, we can represent an assignment as a tuple $\overline{b} \equiv (b_1, \ldots, b_n)$, for $b_i \in \mathbb{B}$, which we call a *Boolean model*. A formula is a *tautology* (*contradiction*) if it is true in all (no) models.

For example, the formula $x_1 \oplus x_2$ defined by $(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$ semantically defines a function that is true in the models $(1,0)$ and $(0,1)$. Alternatively, since a function on $\mathbb{B}^n \to \mathbb{B}$ can also be viewed as a relation on $\mathbb{B}^n$, $\oplus$ defines the relation $\{(1,0),(0,1)\}$. As the standard Boolean connectives are *functionally complete*, every relation over $\mathbb{B}^n$ can be defined by a formula in the logic.

2.1 Example: Gate Level Modeling

Boolean logic can be used to model discrete systems. This is best seen with the example of combinational circuits. Structurally, we can model a circuit C with inputs $i_1, \ldots, i_n$ and outputs $o_1, \ldots, o_m$ by defining m functions on $\mathbb{B}^n \to \mathbb{B}$ that model the computations required to compute each output. By this we mean that the structure of the circuit is reflected in the functions used to compute it, e.g., that each and-gate in the circuit corresponds to an $\wedge$-connective in the function definition.

Note that when we talk about modeling systems, our use of the term *model* is different from the previous semantic use (Boolean model). In the terminology of formal methods practitioners, a model is a construct or mathematical object that describes a system or presents a theory that accounts for properties of a system. Of course, these two uses are connected: a structural or behavioral model has a semantics given by the Boolean model. The role of logic then is to connect these two. By proving that different structural and behavioral models are equivalent, we show that they have the same Boolean models.

Consider a simple example. Figure 1 describes a circuit for a 1-bit full-adder. The two outputs, *sum* and *cout*, are functions of the inputs *a*, *b*, and *cin*. We can model the structure of this circuit in Boolean logic by formulae that reflect how *sum* and *cout* are computed.

$$\begin{aligned} sum(a,b,cin) &\equiv (a \oplus b) \oplus cin \\ cout(a,b,cin) &\equiv (cin \wedge (a \oplus b)) \vee (a \wedge b) \end{aligned}$$

[1] i.e., φ has $x_1, \ldots, x_n$ as variables. For logics with quantifiers, we mean the *free* variables of φ.

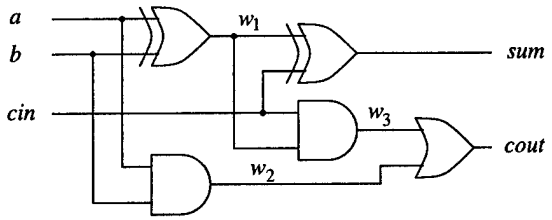


Figure 1. 1-bit full adder

In this structural model, each gate corresponds to a Boolean function and circuit composition is modeled by function composition.

The semantics of these formulae *are* a model of the behavior of the circuit. However, it is often useful to formalize a behavioral model *within* Boolean logic so that we can establish theorems about the equivalence of different structural and behavioral models. For example, we can model the behavior of $sum(a,b,cin)$ by a formula $sum_beh(a,b,cin)$ that enumerates the inputs for which the output bit *sum* is set (*cout_beh* is defined analogously), i.e.,

$$\begin{aligned} &(a \wedge \neg b \wedge \neg cin) \vee (\neg a \wedge b \wedge \neg cin) \vee \\ &(\neg a \wedge \neg b \wedge cin) \vee (a \wedge b \wedge cin)\,. \end{aligned}$$

Although *sum_beh* is logically equivalent to *sum*, it is perhaps more perspicuous as a behavioral description since it amounts to a tabular description of the function computed. Note that the functional completeness of Boolean logic means that every Boolean function can be described by a formula; so Boolean logic is adequate for this kind of behavioral modeling.

Given a formalization of both the structure and the behavior of a circuit in Boolean logic, we can use a decision procedure to validate that the two formalizations are logically equivalent, and thus define identical functions semantically. In this case we verify that

$$\begin{aligned} &(sum(a,b,cin) \leftrightarrow sum_beh(a,b,cin)) \wedge \\ &(cout(a,b,cin) \leftrightarrow cout_beh(a,b,cin))\,. \end{aligned}$$

2.2 Limitations

The above example suggests that, for certain kinds of discrete systems, namely combinational circuits, Boolean logic is adequate to model behavior and structure and to demonstrate their equivalence. However, even for such simple systems, there are inadequacies. In particular, the connection between the model (the formula) and what is modeled (the circuit) may be less direct than we would like. For example, although each

gate in the circuit corresponds to a function in the formula, shared components must be duplicated in the model. So the corresponding formula can be significantly larger and the relationship between subcircuits and subformulae obscured. Moreover, circuits must be modeled as functions when in practice some are better modeled as relations, e.g., bi-directional circuits.

3 Quantified Boolean Logic

A simple extension to Boolean logic is to add the *Boolean quantifiers* $\forall$ and $\exists$; the result is called *quantified Boolean logic* (or QBL). In the semantics, quantifiers are interpreted as ranging over $\mathbb{B}$. The remaining details are identical to Boolean logic.

It follows semantically that we can eliminate quantifiers: in any model $(b_1, \ldots, b_n) \in \mathbb{B}^n$, $\exists x_{n+1} . \varphi(x_1, \ldots, x_n, x_{n+1})$ is semantically equivalent to $\varphi(x_1, \ldots, x_n, 0) \vee \varphi(x_1, \ldots, x_n, 1)$. Replacing $\vee$ with $\wedge$ yields a similar equivalence for $\forall$. We can eliminate quantifiers syntactically too: if we replace 0 (respectively 1) in the above expansion by any suitable contradiction (respectively tautology), then we can use the resulting equivalence to eliminate the Boolean quantifiers in any QBL formula. The resulting formula though may be exponentially larger. This difficulty is reflected in complexity of QBL: it is PSPACE-complete. So, provided that $P \neq NP$, theorem proving for QBL is harder than for Boolean logic.

Although quantifiers do not give QBL any more expressive power than Boolean logic, there is a sense in which QBL is more powerful: some Boolean functions can be expressed more succinctly and, moreover, QBL formulae can better capture the structure of the system modeled. The next example illustrates this.

3.1 Example: Gate Level Modeling

In QBL, we can formalize circuits as relations over their external ports. Circuits are built from relations representing primitives, such as transistors or gates, and are combined with conjunction and 'wired together' using existential quantification. This style of representation is standard in the theorem proving community and scales well to complex systems [6].

For example, to model the adder given in Figure 1 we begin by defining the following gates.

$$\begin{aligned} \mathsf{and}(a, b, o) &\equiv o \leftrightarrow (a \wedge b) \\ \mathsf{or}(a, b, o) &\equiv o \leftrightarrow (a \vee b) \\ \mathsf{xor}(a, b, o) &\equiv o \leftrightarrow ((a \wedge \neg b) \vee (\neg a \wedge b)) \end{aligned}$$

We then compose these to model the adder. The top half of the adder consists of two *xor*-gates, connected by an internal wire w_1, which compute the sum bit *out*. The bottom half uses the internal wire w_1, as well as the two inputs a and b, to compute the carry-out bit *cout*. Our definition conjoins the gate descriptions and projects away the internal wires:

$$\begin{aligned} &\mathsf{full_adder}(a, b, out, cin, cout) \equiv \exists w_1, w_2, w_3 . \\ &\quad \mathsf{xor}(a, b, w_1) \wedge \mathsf{xor}(w_1, cin, out) \wedge \\ &\quad \mathsf{and}(a, b, w_2) \wedge \mathsf{and}(cin, w_1, w_3) \wedge \mathsf{or}(w_3, w_2, cout) \end{aligned}$$

This formula is a better structural model of the circuit pictured in Figure 1 than the previous one in Boolean logic. Of course, it is still only a model and it abstracts away aspects of actual circuits such as layout. This is acceptable when we want to reason about functional behavior at the level of bits, but unacceptable for other properties, e.g., delay. Modeling other aspects requires either formalizing them in QBL or a richer logic.

3.2 Limitations

Although QBL offers advantages over Boolean logic, many kinds of discrete systems cannot be modeled in it. In our example, we modeled the *static behavior* of a system, i.e., the relationship between its inputs and outputs, which constitute the *system state*. However, it is difficult to model *dynamic behavior*, that is, how the state changes over time.

We can only approximate this. Namely, rather than formalizing a predicate φ of the state variables $\overline{x}$, we instead formalize a Boolean 'before-after' predicate $\varphi'(\overline{x}, \overline{x}')$ that contains a 'primed' version x' of each state variable x and formalizes how the state changes over one unit of time. In this way we can formalize dynamic behavior over any fixed number of time intervals; however formalizing behavior over arbitrary time intervals is impossible.

A second limitation is that these logics cannot be used to model families of related systems. For example, our 1-bit adder specifications can be generalized to adders over 2, 3, ... bit words. Hardware description languages such as VHDL [1] typically provide some kind of parameterization mechanism which, combined with iteration, allows parametric descriptions. Again, this is impossible using the weak logics that we have so far considered.

It turns out that a generalization of the approximation idea described above can be used, in many cases, to solve both problems. Rather than writing a formula $\varphi(\overline{x})$ with models $\overline{b} \in \mathbb{B}^n$, we instead let $\overline{x}$ range over sequences of Boolean models $\overline{b}(0)\overline{b}(1)\ldots\overline{b}(m)$ of arbitrary, but finite, length. That is, a model is a string that states how the values (Boolean model) of the n

state variables evolve over m time intervals. Alternatively, the model can represent the behavior of a system with n state variables, each ranging over values of an arbitrary 'length' m. This generalization requires generalizing QBL, which is a logic of bits, to a logic of bit sequences, or strings.

4 Monadic Logic of Strings

We present here a monadic logic of strings called WS1S. Work on WS1S and related monadic theories goes back to Büchi [5], Elgot [8], and Trakhtenbrot [18]. We briefly review syntax and semantics.

Let x and X range over disjoint sets $\mathcal{V}_1$ and $\mathcal{V}_2$ of first and second-order variables. The language of WS1S is described by the following grammar.

$$t ::= x \mid 0 \mid \mathsf{s}(t)$$
$$\varphi ::= X(t) \mid \varphi \wedge \varphi \mid \neg\varphi \mid \exists x.\varphi \mid \exists X.\varphi$$

Hence terms are built from first-order variables, the constant 0, and the successor symbol. Formulae are built from atoms $X(t)$ and are closed under conjunction, negation, and quantification over first and second-order variables. Other connectives and quantifiers can be defined using standard classical equivalences, e.g., $\forall x.\varphi \equiv \neg\exists x.\neg\varphi$.

Semantically, WS1S formulae are interpreted in $\mathbb{N}$. 0 and s are zero and the successor function and $X(t)$ is true when the number denoted by t is in the set of numbers denoted by X. First-order quantification is quantification over natural numbers, whereas second-order quantification is quantification over *finite* sets of natural numbers.

WS1S stands for the *weak second-order monadic theory of one successor.* It is monadic as all atoms are unary predicates. It is weak because second-order variables are interpreted over finite (instead of arbitrary subsets) of the domain. And it has only a single successor function.

Although WS1S is a logic of numbers and sets, we motivated it as a logic of strings, as this emphasizes that WS1S is a natural extension of (quantified) Boolean logic.[2] The relationship is as follows. Any finite string $b(0)b(1)\ldots b(m)$ over $\mathbb{B}$ encodes a finite set of positions, namely $\{p \in \{0,\ldots,m\} \mid b(p) = 1\}$. More generally, we can encode n strings over $\mathbb{B}$ as a single string over $\mathbb{B}^n$. Hence, if $\varphi(\overline{X})$ is a formula whose free second-order variables are $\overline{X} \equiv X_1,\ldots,X_n$, a WS1S *model* can be encoded by a finite string over the alphabet $\mathbb{B}^n$.

As a simple example, the formula $\varphi(X,Y)$ given by

$$X(0) \wedge \forall p.\,(X(p) \leftrightarrow Y(\mathsf{s}(p)))$$

states that the string X begins with a 1 and that Y equals X shifted one position to the right. A model for this formula is a string $\overline{b}(0)\overline{b}(1)\ldots\overline{b}(m)$ where each $\overline{b}(i)$ is a letter in $\mathbb{B}^2$. To visualize this, we write letters (b_1,b_2) vertically; the first row encoded in the string determines an interpretation for X, and the second for Y. Two such models are

X	1	0	1	0	and	X	0	1	0
Y	1	1	0	1		Y	0	0	0

The first model, for example, interprets X as $\{0,2\}$ and Y as $\{0,1,3\}$. φ is true in the first model and false in the second.

Since models are strings, the set of models that satisfy a formula constitutes a language. A fundamental theorem for WS1S is that this logic precisely captures the regular languages: the language associated with each WS1S formula φ is regular, and vice versa. This is the key to decidability. Using automata theoretic techniques, given a formula φ, we can construct an automaton that recognizes the models that satisfy φ. A formula $\varphi(\overline{X})$ is a tautology when the corresponding automaton accepts the universal language on $\mathbb{B}^n$. The 'vice versa' part of this theorem gives us an analog of functional completeness for WS1S: we can model the behavior of any system whose behavior is regular in the language theoretic sense.

WS1S is substantially more expressive than QBL, but this comes at a high price. WS1S is nonelementary decidable [12]: a formula of size n may require time and space bounded below by an iterated stack of exponentials whose height is proportional to n.

4.1 Tools for WS1S

Despite the high complexity of WS1S, several groups have implemented decision procedures for it [10, 11, 13] that work surprisingly well on many nontrivial problems. All examples in this paper have been run in the MONA system [10], which implements a decision procedure for WS1S and its generalization to trees considered in the following section.

MONA offers various kinds of syntactic sugar that are useful in large specifications, which we employ in our examples. For example, first-order variables are actually defined in terms of second-order variables (where a first-order variable is interpreted by a string with a

[2] This view is also taken by Thomas in [16] where Boolean logic is used to motivate a closely related monadic logic of strings, which he calls Büchi-Elgot-Trakhtenbrot logic.

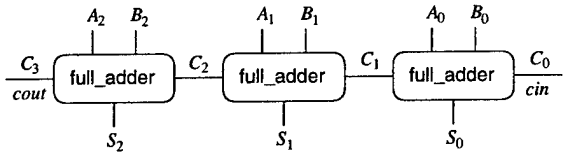


Figure 2. An n-bit adder instance, for $n = 3$

1 in only a single position, giving the value of the first-order term). Hence formulae may have free variables of either order. We also employ sugared notation for QBL, which is definable in WS1S by reasoning about the 0th position of second-order variables. For example, the QBL formula $\forall p.\, \exists q.\, p \leftrightarrow q$ can be defined by the WS1S formula $\forall P.\, \exists Q.\, P(0) \leftrightarrow Q(0)$. MONA supports the use of Boolean variables and Boolean quantifiers. To distinguish between Boolean, first, and second-order variables we use the following naming convention: Second-order variables are capitalized, first-order variables are lower-case letters with names like i, p, and q, and Booleans are lower-case strings like a, b, *cin* and *cout*. Moreover, we superscript quantifiers with either a 0, 1, or 2 to indicate Boolean, first, and second-order quantification respectively.[3] Finally from the minimal syntax of WS1S we have given, one can define additional operators with their expected meaning, e.g., comparison operators on terms like $=$ and $\leq$. We use such operators freely.

4.2 Example: Parameterized Ripple-Carry Adder

Here we give a simple example of how WS1S can be used to structurally and behaviorally model a parameterized family of circuits. For more sophisticated examples, as well as examples of modeling dynamic (sequential) systems, see [4].

Let us start with a structural model. We have previously shown how to model a 1-bit adder. Figure 2 suggests how an adder can be iterated, in a 'ripple-carry' way, to construct an n-bit adder, for $n = 3$. In general, we can model an n-bit adder by wiring together n 1-bit adders where the carry-out of the ith adder is the carry-in of the i+1st. The first carry has the value of the carry-in and the last has the value of the carry-out.

It is easy to formalize a ripple-carry architecture by a WS1S formula. If we use C to represent the sequence of carries, we can formalize the general case by the following formula, which relates three strings (the inputs A and B and the output S) and two Booleans (the carry-in *cin* and carry-out *cout*); the number of bits added is given by the parameter n.

$$\begin{aligned}&\mathsf{adder}(n, A, B, S, cin, cout) \equiv \\ &\quad \exists^2 C.\, (C(0) \leftrightarrow cin) \wedge (C(n) \leftrightarrow cout) \wedge \\ &\qquad \forall^1 p.\, p < n \rightarrow \\ &\qquad\quad \mathsf{full_adder}(A(p), B(p), S(p), C(p), C(p+1))\end{aligned}$$

This is a direct formalization of our natural language description where quantification over positions p formalizes iteration (generalized conjunction) over the n 1-bit adders. Note that given the encoding of Boolean logic in WS1S, we can use the previously given structural model of a 1-bit full adder.

Let us now turn our attention to a behavioral model in which we state how strings, representing n-bit binary numbers, are added. WS1S is a logic of strings and arithmetic must be encoded within its limited language. We model addition by formalizing the standard algorithm for adding base-two numbers: The ith output bit is set if the sum of the ith inputs and carry-in is 1 mod 2, and the ith carry bit is set if at least two of the previous inputs and carry-in are set. Since we specify the behavior of an n-bit adder, we must restrict the addition modulo 2^n and compute the carry values as special cases.

$$\begin{aligned}\mathsf{at_least_two}(a, b, c, d) &\equiv d \leftrightarrow (a \wedge b) \vee (a \wedge c) \vee (b \wedge c) \\ \mathsf{mod_two}(a, b, c, d) &\equiv a \leftrightarrow b \leftrightarrow c \leftrightarrow d\end{aligned}$$

$$\begin{aligned}&\mathsf{adder_beh}(n, A, B, S, cin, cout) \equiv \\ &\quad \exists^2 C.\, (C(0) \leftrightarrow cin) \wedge (C(n) \leftrightarrow cout) \\ &\qquad \forall^1 p.\, p < n \rightarrow \\ &\qquad\quad \mathsf{at_least_two}(A(p), B(p), C(p), C(p+1)) \\ &\qquad\quad \mathsf{mod_two}(A(p), B(p), S(p), C(p))\end{aligned}$$

This specifies a language over $\mathbb{B}^6$, which encodes interpretations for n, A, B, S, *cin* and *cout*. For example, one string in this language is:

	0	1	2	3	4
n	0	0	0	0	1
A	1	1	0	0	0
B	1	0	0	1	0
S	1	0	1	1	0
cin	1	0	0	0	0
cout	0	0	0	0	0

(1)

Recall that in the interpreting string for first-order variables like n, exactly the nth bit is 1. Moreover, *cin* and *cout* are true iff the 0th bit of their interpreting string is 1. Hence the above encodes that the bit-width $n = 4$. Reading Boolean strings as binary numbers from left to right (up to the nth position) the next three lines encode $A = 3$, $B = 9$, and $S = 13$. Since the carry-in

[3] Any remaining ambiguity (indeed, even without any of these conventions) can be resolved from context.

is set and the carry-out isn't, this string does is indeed in the addition relation modeled.

The adder's structural and behavioral models are similar in that they both iterate components that add a single digit. Using similar models may be natural, but it runs the risk of making similar mistakes in both.

An alternative behavioral model that avoids this problem is based on the equation

$$val(n,S) + 2^n vl(cout) = val(n,A) + val(n,B) + vl(cin)\,, \tag{2}$$

where $val(n,S)$ computes the value of a string S as an n-bit word, and vl converts a Boolean to a number. With this model we define once and for all an encoding of natural numbers as strings, together with predicates encoding (Presburger) arithmetic operations. This encoding of arithmetic can be reused when specifying the behavior of other circuits, like counters, comparators and the like.

$$\begin{aligned}
\mathsf{val}(n,X,Y) &\equiv \forall^1 p.\, Y(p) \leftrightarrow (p < n \wedge X(p)) \\
\mathsf{powof2}(n,b,X) &\equiv \forall^1 p.\, X(p) \leftrightarrow (p = n \wedge b) \\
\mathsf{vl}(b,N) &\equiv \forall^1 p.\, N(p) \leftrightarrow (p = 0 \wedge b) \\
\mathsf{add}(A,B,S) &\equiv \exists^2 C.\, \neg C(0) \wedge \\
&\quad \forall^1 p.\, \mathsf{at_least_two}(A(p),B(p),C(p),C(p+1)) \\
&\quad \wedge \mathsf{mod_two}(A(p),B(p),S(p),C(p))
\end{aligned}$$

We now directly encode (2) as

$$\begin{aligned}
&\mathsf{adder_beh}(n,A,B,S,cin,cout) \equiv \\
&\exists^2 A',B',S',CIN,COUT,U,V,W. \\
&\qquad \mathsf{vl}(cin,CIN) \wedge \mathsf{powof2}(n,cout,COUT) \\
&\qquad \wedge \mathsf{val}(n,A,A') \wedge \mathsf{val}(n,B,B') \\
&\qquad \wedge \mathsf{val}(n,S,S') \\
&\qquad \wedge \mathsf{add}(S',COUT,U) \wedge \mathsf{add}(A',B',V) \\
&\qquad \wedge \mathsf{add}(V,CIN,W) \wedge U = W\,.
\end{aligned}$$

4.3 Verification

We now have both a structural model and two equivalent (and this equivalence has been automatically checked by MONA), but differently motivated, behavioral models for a family of n-bit adders. We can formalize the correctness of the structural model, with respect to either behavioral model, as follows.

$$\begin{aligned}
&\forall^1 n.\, \forall^2 A,B,S.\, \forall^0 cin,cout. \\
&\quad \mathsf{adder}(n,A,B,S,cin,cout) \leftrightarrow \\
&\quad \mathsf{adder_beh}(n,A,B,S,cin,cout)
\end{aligned}$$

This is verified by MONA in under a second on a Sun Ultra-Sparc workstation. This tells us that any adder in this family, e.g., a 64-bit adder, will operate as expected, provided the gates it is built from function as specified. Such a parameterized statement cannot be expressed in QBL.

4.4 Limitations

Any structural or behavioral model formalized in WS1S defines a regular language over a fixed alphabet $\mathbb{B}^n$. This limits what can be modeled.

To begin with, observe that there is an asymmetry: given a formula, the alphabet size and thus the 'width' of any string, is fixed, while the length of any string is finite, but a priori unbounded. As motivated in Section 3.2, we can use the one 'unbounded dimension' to model the dynamic (sequential) behavior of systems or, alternatively, families of related systems, e.g., parameterized by their bus-width. However, we cannot overcome this asymmetry and model systems where both the width and length dimension are unbounded, i.e., problems that can be modeled on two-dimensional grids instead of one-dimensional strings.[4] As a result, many practical problems concerning discrete systems cannot be modeled. For example, we cannot reason about families of sequential systems such as n-bit registers or counters.

Furthermore, we can only model systems with regular behavior. A parameterized adder defines a regular language; the intuition for this is that an automaton can read a string of bits like that in (1) and it need only propagate bounded information across each letter (e.g., information about the current carry) to determine if a string is in the language. However not all parameterized systems have this property. For example, the language associated with a family of n-bit multipliers is not regular.

Despite these limitations, WS1S is quite expressive. Since every regular language can be represented by a WS1S formula, any discrete system whose behavior is regular, or that can be structurally modeled as a finite state transition system, can be formalized in WS1S. And this covers many systems considered in practice. Furthermore, in some cases, we can model two-dimensional systems that use the second dimension in a limited way. For example, those systems that correspond not to problems on grids, but to problems on trees. Such systems play an important role, e.g., in designing efficient circuits using divide and conquer techniques.

5 Monadic Logic of Trees

In Section 4 we modeled an n-bit ripple-carry adder by cascading n adders. This is correct although inefficient since a delay, linear in n, is required to propagate

[4] Modeling problems on grids requires a stronger logic, e.g., with dyadic predicates. This leads to undecidability as one can encode problems about Turing machine computations in them.

the carry bits between stages. An alternative design is a tree-structured circuit with a logarithmic delay.

We cannot model tree-like systems using a logic of strings. Fortunately, there is a simple generalization that suffices. Rather than having a single successor for the next position in the string, we have two successors, for 'left' and 'right'. Then, just as finite sets of positions over one successor are encoded by strings, sets of positions over two successors are encoded by binary trees. Working out the details yields the *weak second-order monadic theory of two successors* (or WS2S) [9, 15, 17].

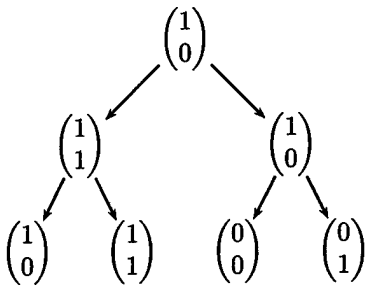
The syntax of WS2S is similar to WS1S except we use ε (instead of 0) to denote the root address in a tree and we now have two successor .0 (left child) and .1 (right child). Semantically, WS2S formulae are interpreted in the domain $\mathcal{D} = \{0,1\}^\star$ and ε denotes the empty string, .0 denotes concatenation with 0, and .1 denotes concatenation with 1. All other details are identical to WS1S.

WS2S generalizes Boolean logic to a logic of trees because second-order variables are interpreted by finite subsets $M \subset \mathcal{D}$ and any such M can be encoded by a Boolean-labeled, binary branching tree. The decidability of this logic follows by generalizing the decision procedure for WS1S: the models of any formula $\varphi(x_1, \ldots, x_n)$ define a regular tree language over the alphabet $\mathbb{B}^n$ and given φ we can construct a tree automaton accepting this language. As with WS1S, WS2S is non-elementary.

The following is a simple example of a specification in WS2S.

$$X(\varepsilon) \wedge (\forall^1 p.\, X(p.0) \leftrightarrow X(p.1)) \wedge \forall^1 p.\, \neg Y(p.0) \vee \neg Y(p.1)$$

This states that X contains the root position ε and that a position p is in X iff its brother is also in X. Moreover, for any p, Y contains at most one of p's successors. The interpretation $\{\varepsilon, 0, 1, 00, 01\}$ for X and $\{0, 01, 11\}$ for Y satisfies this formula and is encoded by the following tree (where the first component of each pair encodes the interpretation of X and the second encodes Y).



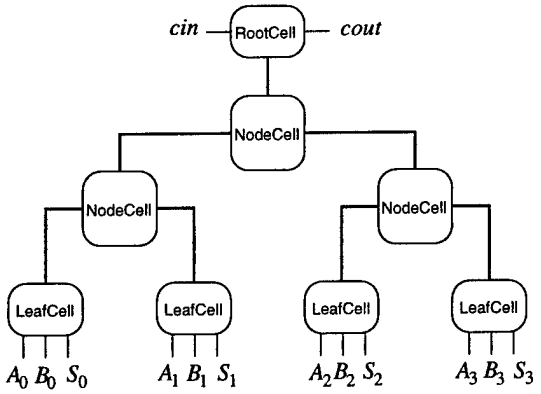


Figure 3. An n-bit CLA instance, for $n = 4$

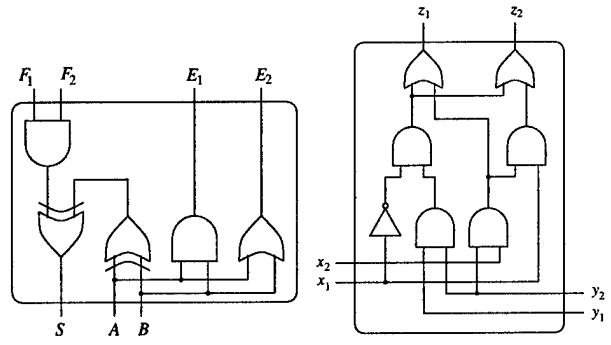
5.1 Example: Parameterized Carry-Lookahead Adder

We begin by describing a family of n-bit *carry-lookahead adders* (or CLAs) whose overall structure is given (for $n = 4$) in Figure 3. We do not describe the adder in detail (see [7]) and restrict ourselves to a few comments. The adder operates in two phases: an *upward* phase and *downward* phase. In the upward phase, a carry status is computed for each internal node. This indicates whether an incoming carry is *killed*, *propagated*, or a new carry is *generated*. The carry status is given by the wires E_1 and E_2. At the leafs of the tree, the carry status for each digit is computed using the circuit depicted in Figure 4(a). At inner nodes, an operator $\otimes$ is used to combine the carry statuses of the successor nodes (see Figures 4(d) and 4(b)).

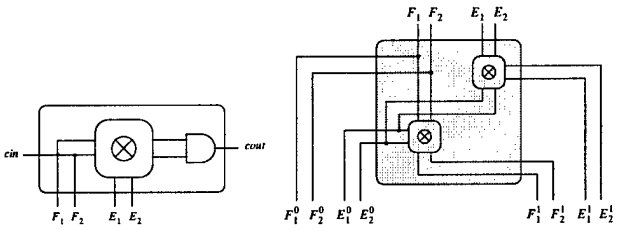
In the downward phase, an inner node passes the incoming carry bits F_1 and F_2 unchanged as signals F_1^0 and F_2^0 to its left successor; the carry bits F_1^1, F_2^1 for the right successor are computed using the operator $\otimes$ from the carry statuses and the incoming carry bits (see Figure 4(d)). As indicated in Figure 4(c), the incoming carry at the root is given by *cin* and the outgoing carry by *cout*. At the leaf nodes, the incoming carry bits F_1 and F_2 are used to compute the sum bit as depicted in Figure 4(a).

5.2 Structural Modeling of the CLA

Structural modeling in WS2S is identical to WS1S except now we use two successors to index into trees. Hence we translate the circuits in Figures 4(a)–4(d) directly into the following WS2S formulae. Note that LeafCell and NodeCell take an extra argument p that indicates which leaf or node values are used.



(a) LeafCell Circuit (b) ⊗ Circuit



(c) RootCell Circuit (d) NodeCell Circuit (op)

Figure 4. Components of the CLA Circuit

$$\mathsf{LeafCell}(A,B,S,F_1,F_2,E_1,E_2,p) \equiv$$
$$\mathsf{and}(A(p),B(p),E_1(p)) \wedge \mathsf{or}(A(p),B(p),E_2(p)) \wedge$$
$$\exists^0 w_1,w_2.\,\mathsf{and}(F_1(p),F_2(p),w_1)) \wedge$$
$$\mathsf{xor}(A(p),B(p),w_2) \wedge \mathsf{xor}(w_1,w_2,S(p))$$

$$\mathsf{NodeCell}(F_1,F_2,E_1,E_2,p) \equiv$$
$$(F_1(p.0) \leftrightarrow F_1(p)) \wedge (F_2(p.0) \leftrightarrow F_2(p)) \wedge$$
$$\mathsf{op}(F_1(p),F_2(p),E_1(p.0),E_2(p.0),F_1(p.1),F_2(p.0)) \wedge$$
$$\mathsf{op}(E_1(p.0),E_2(p.0),E_1(p.1),E_2(p.1),E_1(p),E_2(p))$$

$$\mathsf{RootCell}(F_1,F_2,E_1,E_2,cin,cout) \equiv$$
$$(cin \leftrightarrow F_1(\varepsilon)) \wedge (cin \leftrightarrow F_2(\varepsilon)) \wedge$$
$$(\exists^0 a,b.\,\mathsf{op}(F_1(\varepsilon),F_2(\varepsilon),E_1(\varepsilon),E_2(\varepsilon),a,b) \wedge$$
$$\mathsf{and}(a,b,cout))$$

$$\mathsf{op}(x_1,x_2,y_1,y_2,z_1,z_2) \equiv \exists^0 w_1,w_2,w_3,w_4,w_5.$$
$$\mathsf{not}(x_1,w_1) \wedge \mathsf{and}(y_1,y_2,w_2) \wedge \mathsf{and}(x_2,y_2,w_3) \wedge$$
$$\mathsf{and}(w_1,w_2,w_4) \wedge \mathsf{and}(x_1,w_3,w_5) \wedge$$
$$\mathsf{or}(w_4,w_5,z_1) \wedge \mathsf{or}(w_4,w_3,z_2)$$

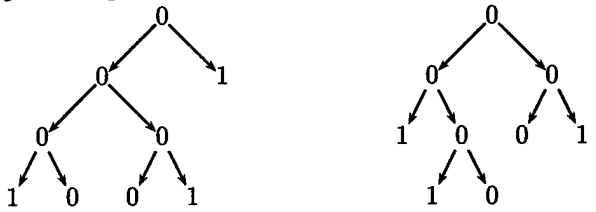
The overall circuit, given in Figure 3, is modeled by the predicate

$$\mathsf{cla}(A,B,S,E_1,E_2,F_1,F_2,cin,cout) \equiv \exists^2 T.$$
$$\mathsf{RootCell}(F_1,F_2,E_1,E_2,cin,cout) \wedge$$
$$(\forall^1 p.\,(\mathsf{leaf}(p,T) \rightarrow \mathsf{LeafCell}(A,B,S,F_1,F_2,E_1,E_2,p)) \wedge$$
$$(\mathsf{node}(p,T) \rightarrow \mathsf{NodeCell}(F_1,F_2,E_1,E_2,p))) \wedge$$
$$\mathsf{shape_cond}(A,B,S,T).$$

RootCell initializes the carry-in bit and computes the carry-out bit. The next two lines correspond to a for-loop with discrimination (pattern matching) for each position p. The first implication gives the base case: each leaf of the circuit is LeafCell. The second gives the step case: each inner node of the circuit is NodeCell. The predicate shape_cond fixes the shape of the inputs; we explain it shortly.

5.3 Behavior

In WS1S we encoded numbers as bit-strings. In WS2S we have binary trees and encode numbers using the labels on a tree's frontier. For example, the following trees represent the bit-strings 10011 and 11001.



An important requirement in specifying tree structured adders, is that, once we fix the format of the adder to be a particular shape, both inputs and the output must also be of that shape; that is, numbers must be encoded at leaf nodes in the same positions in these three trees. This kind of requirement, which corresponds roughly to a kind of 'type-correctness' for the inputs, is easy to specify in a high-level programming or specification language where recursive data-types can formalize this type (or 'shape') constraint. In our setting, we define a predicate (shape_cond, which we used in the specification of cla above) that enforces this requirement.

We proceed in several steps. First, we characterize those trees with a particular shape as those T where:

1. T is not emtpy.
2. T is closed under the parent relation: if a position p is in T then its parent, denoted by $p\uparrow$, is in also in T.
3. If an inner node p is in T then its both successors $p.0$ and $p.1$ are in T too.

This 'type' is formalized by

$$\mathsf{shape}(T) \equiv (\exists^1 p.\,T(p)) \wedge$$
$$\forall^1 p.\,(T(p) \rightarrow T(p\uparrow)) \wedge (T(p) \wedge T(p.0) \rightarrow T(p.1)) \wedge$$
$$(T(p) \wedge T(p.1) \rightarrow T(p.0)).$$

Second, we say that a tree X has shape T (X is of type T), if all its positions constitutes a subset of the leafs of T.

$$\mathsf{is_of_shape}(X,T) \equiv \forall^1 p.\, X(p) \rightarrow \mathsf{leaf}(p,T).$$

Finally, we combine these to formalize shape_cond that holds when the trees A, B, and S, which represent base-two numbers, have the same shape T.

$$\begin{aligned}\mathsf{shape_cond}(A,B,S,T) \equiv\ & \mathsf{shape}(T) \ \wedge\ \mathsf{is_of_shape}(A,T) \ \wedge \\ & \mathsf{is_of_shape}(B,T) \ \wedge\ \mathsf{is_of_shape}(S,T).\end{aligned}$$

To complete our behavioral model, we define several auxiliary predicates in Figure 5 that allow us to traverse the leaves of a valid input tree from left to right. For a tree T satisfying the shape predicate and a position p, $\mathsf{first}(p,T)$ checks if p is the left-most leaf in T and $\mathsf{last}(p,T)$ checks if p is the right-most leaf in T. The predicate $\mathsf{next}(p,q,T)$ checks if p and q are leaves in T and q is the next leaf to the right of q. Using these, our behavioral model can be defined analogously to the first behavioral model in Section 4.

$$\begin{aligned}&\mathsf{adder_beh}(A,B,S,cin,cout) \equiv \exists^2 T, C.\\ &\quad \mathsf{shape_cond}(A,B,S,T) \ \wedge\\ &\quad \forall^1 p.\, \mathsf{leaf}(p,T) \rightarrow \mathsf{mod_two}(A(p),B(p),C(p),S(p)) \ \wedge\\ &\quad \exists^1 p.\, \mathsf{first}(p,T) \ \wedge\ (cin \leftrightarrow C(p)) \ \wedge\\ &\quad \exists^1 p.\, \mathsf{last}(p,T) \ \wedge\ \mathsf{at_least_two}(A(p),B(p),C(p),cout) \ \wedge\\ &\quad \forall^1 p,q.\, \mathsf{leaf}(p,T) \ \wedge\ \mathsf{next}(p,q,T) \rightarrow\\ &\qquad\qquad \mathsf{at_least_two}(A(p),B(p),C(p),C(q))\ .\end{aligned}$$

5.4 Verification

We can now verify that our structural model of the CLA is equivalent to our behavioral model, i.e., that a CLA of any size actually adds its inputs. We formalize this as

$$\begin{aligned}&\forall^2 A,B,S.\, \forall^0 cin, cout.\\ &\ \mathsf{cla}(A,B,S,cin,cout) \leftrightarrow \mathsf{adder_beh}(A,B,S,cin,cout)\ .\end{aligned}$$

This formula is proved by MONA in one second.

6 Conclusion and Prospects

Monadic logics have been studied by logicians for over forty years, but are almost unknown to engineers. We have argued that these logics are natural generalizations of Boolean logic that can be motivated from an engineering perspective and used to model many more kinds of systems. They are decidable, which means that verification can be automated, and they have an operational side as formulae correspond to automata over strings or trees. This opens possibilities for designing modeling environments that support a variety of formal activities including verification, simulation, and testing.

$$\mathsf{leaf}(p,T) \equiv T(p) \ \wedge\ \neg T(p.0) \ \wedge\ \neg T(p.1)$$

$$\mathsf{node}(p,T) \equiv T(p) \ \wedge\ T(p.0) \ \wedge\ T(p.1)$$

$$\begin{aligned}\mathsf{path}(p,X) \equiv\ & \mathsf{leaf}(p,X) \ \wedge\\ & \forall^1 x.\, (X(x) \rightarrow X(x\uparrow)) \ \wedge\ (X(x.0) \rightarrow \neg X(x.1)) \ \wedge\\ & \quad (X(x.1) \rightarrow \neg X(x.0))\end{aligned}$$

$$\begin{aligned}\mathsf{next}(p,q,T) \equiv\ & p \neq q \ \wedge\ \mathsf{leaf}(p,T) \ \wedge\ \mathsf{leaf}(q,T) \ \wedge\\ & \exists^2 P,Q.\, \mathsf{path}(p,P) \ \wedge\ \mathsf{path}(q,Q) \ \wedge\\ & \exists^1 s.\, \exists^2 S.\, P(s) \ \wedge\ Q(s) \ \wedge\ P(s.0) \ \wedge\\ & \quad Q(s.1) \ \wedge\ \mathsf{path}(s,S) \ \wedge\\ & \forall^1 u.\, (P(u.0) \ \wedge\ u \neq s \rightarrow S(u.0)) \ \wedge\\ & \quad (Q(u.1) \ \wedge\ u \neq s \rightarrow S(u.1))\end{aligned}$$

$$\begin{aligned}\mathsf{first}(p,T) \equiv\ & \mathsf{leaf}(p,T) \ \wedge\ \exists^2 X.\, \mathsf{path}(p,X) \ \wedge\\ & \forall^1 u.\, X(u.1) \rightarrow (\forall^1 v.\, u.0 \leq v \rightarrow \neg T(v))\end{aligned}$$

$$\begin{aligned}\mathsf{last}(p,T) \equiv\ & \mathsf{leaf}(p,T) \ \wedge\ \exists^2 X.\, \mathsf{path}(p,X) \ \wedge\\ & \forall^1 u.\, X(u.0) \rightarrow (\forall^1 v.\, u.1 \leq v \rightarrow \neg T(v))\end{aligned}$$

Figure 5. Auxiliary predicates for the CLA

One drawback is that, in some cases, problems must be modeled at too low a level of abstraction: in WS2S everything must be mapped onto binary branching Boolean trees. This can make models difficult to design and understand (witness the concern with 'shape constraints' in Section 4.2) and lead to modeling errors.

Our current work aims at building a high-level specification language that eliminates this drawback. We have developed a language that allows users to specify tree languages in WS2S using data-types like those found in modern programming languages [2]. Moreover, we are building a translator from the high-level hardware description language VHDL to WS1S and integrating the MONA system with other development and verification tools [3].

References

[1] ANSI/IEEE Standard 1076–1993. *IEEE Standard VHDL Language Reference Manual.* IEEE, New York, USA, June 1994.

[2] A. Ayari, D. Basin, and A. Podelski. Lisa: A Specification Language Based on WS2S. In *CSL'97*, volume 1414 of *LNCS*. Springer, 1997.

[3] D. Basin and S. Friedrich. Combining WS1S and HOL. In *Frontiers of Combining Systems, Second International Workshop, Amsterdam, September 1998*, Applied Logic Series. Kluwer Academic Publishers, 1998. To appear.

[4] D. Basin and N. Klarlund. Automata based symbolic reasoning in hardware verification. *The Journal of Formal Methods in Systems Design*, 13(3):255–288, November 1998.

[5] J. R. Büchi. Weak second-order arithmetic and finite automata. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 6:66–92, 1960.

[6] A. J. Camilleri, M. J. C. Gordon, and T. F. Melham. Hardware verification using higher-order logic. In D. Borrione, editor, *From HDL Descriptions to Guaranteed Correct Circuit Designs*. North Holland, 1986.

[7] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms*. MIT Press and McGraw-Hill Book Company, 6th edition, 1992.

[8] C. C. Elgot. Decision problems of finite automata design and related arithmetics. *Transactions of the AMS*, 98:21–52, 1961.

[9] Y. Gurevich. Monadic second-order theories. In J. Barwise and S. Feferman, editors, *Model-Theoretic Logics*, pages 479–506. Springer, 1985.

[10] N. Klarlund and A. Møller. *MONA Version 1.3 User Manual*. BRICS Notes Series NS-98-3 (second revision), Department of Computer Science, University of Aarhus, October 1998.

[11] Z. Manna et al. STeP: The Stanford Temporal Prover. In M. Mosses, M. Nielsen, and M. Schwartzbach, editors, *TAPSOFT '95: Theory and Practice of Software Development, 6th International Joint Conference CAAP/FASE*, volume 915 of *LNCS*, pages 793–794. Springer, 1995.

[12] A. Meyer. Weak monadic second-order theory of successor is not elementary-recursive. In *LOGCOLLOQ: Logic Colloquium*. Lecture Notes in Mathematics, No. 453, Springer-Verlag, 1975.

[13] F. Morawietz and T. Cornell. On the recognizibility of relations over a tree definable in a monadic second-order tree description language. Research Report SFB 340-Report 85, Sonderforschungsbereich 340 of the Deutsche Forschungsgemeinschaft, February 1997.

[14] M. O. Rabin and D. Scott. Finite automata and their decision problems. In E. F. Moore, editor, *Sequential Machines: Selected Papers*, pages 63–91. Addison-Wesley, Reading, MA, 1964.

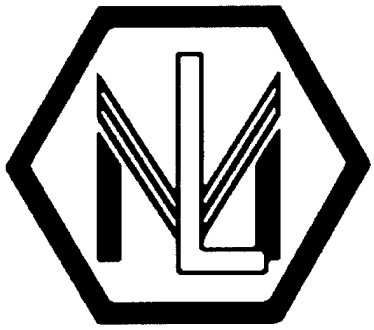
[15] J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem in second-order logic. *Math. Systems Theory*, 2:57–81, 1968.

[16] W. Thomas. Die Logiken von Boole und Büchi-Elgot-Thraktenbrot in der Beschreibung diskreter Systeme. Manuskript, Lehrstuhl für Informatik VII, RWTH Aachen.

[17] W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 4, pages 133–191. Elsevier Science Publishers B. V., 1990.

[18] B. A. Trakhtenbrot. Finite automata and the logic of one-place predicates. *Sib. Math. J.*, 3:103–131, 1962.

Session VIa
Decision Diagrams
Chair: Rolf Drechsler



Matrix-Valued EXOR-TDDs in Decomposition of Switching Functions

Radomir S. Stanković
Braće Taskovića 17/29
18000 Niš
Yugoslavia

Abstract

EXOR Ternary Decision Diagrams (EXOR-TDDs) proved useful in several applications in logic design. This paper shows that EXOR-TDDs are also useful in functional decomposition. We prove that checking simple disjoint decomposition of f with respect to a subset of r variables is equivalent to building the EXOR-TDD for a matrix-valued function of $(n-r)$ variables. Transferring the problem of decomposition into the problem of building EXOR-TDDs permits formulation of a simple decomposition procedure. The same method applies to simple disjoint bi-decomposition of switching functions.

1. Introduction

Decomposition of switching functions is an important task, since when decomposition is possible, it leads to many advantages in network synthesis [10], [12]. At the same time, this is a difficult task. For example, simple disjoint decomposition is a restricted class of functional decompositions defined as follows.

Definition 1: An n-variable switching function f is simple disjoint decomposable with respect to the set of first r variables iff it can be represented as

$$f(x_1,\ldots,x_n) = g(h(x_1,\ldots,x_r), x_{r+1},\ldots,x_n), \qquad (1)$$

where g and h are switching functions of r and $(n-r)$ variables, respectively.

For a given function, to disclose even such decomposition is both time and space consuming.

The relation (1) expresses decomposition with respect to the first r variables in f. We assume that we have developed a procedure to check and perform such decomposition for a given f, if the decomposition is possible.

The same decomposition procedure can be used to check decomposability with respect to any other subset of r variables $x_{j1},\ldots,x_{jr}$, if instead f we consider an auxiliary function f' derived from f by the permutation of variables $x_{j1},\ldots,x_{jr}$ and $x_1,\ldots,x_r$.

Decomposition of a given f is possible iff there are some relationships among the values of f. In general, these relationships reduce to the requirement that there are some isomorphic parts in f. For a chosen representation for f, these isomorphic parts may be properly defined and specified. For example, if decomposition is performed through a decomposition chart, we should check equality or otherwise of some columns, respectively rows, in the chart [2]. If f is given by some analytic expressions as Positive Polarity Reed-Muller expressions, or Sum-of-Products expressions, we should check whether literals in product terms belong to some predetermined subsets of variables [15]. If f is given by the truth-vector, we should check equality of some subvectors in it. If decomposition of f is performed through a decision diagram for f, we should check equality of some subtrees in the decision diagram [3], [10].

There are many decomposition methods for different definitions of the decompositions proposed in the literature from the first considerations of the decomposability [2], up to the related recent publications [3], [4], [7], [8], [9], [10], [13], [15], [17]. Different methods are based on different representations of switching functions, attempting to take advantages from peculiar properties of these, possibly reduced or compact, representations. However, hardly any of these methods exceeds exhaustive search for, in some sense, isomorphic or equal parts in some representation for f.

Present decomposition methods start from the existing representations for switching functions. These representations are intended to provide as compact representations for f as possible, or are adapted to some particular applications. The decomposition methods are directed towards algorithms for efficient manipulations with these representations to check decomposability.

A proper solution of the decomposition problem would be to define purposely a data structure for representation of switching functions, where criteria for decomposability in a given form would be apparent. However, this is hardly possible, at least it is not easy within the algebraic structures usually used in switching theory. A reason is that in these structures, relationships among f and decomposition parts h and g of f, are complicated. Therefore, it is not easy to define a data structure which at the same time ensure a compact representation for f and where the decomposability criterion is simply expressed.

In this paper, we are trying to follow a way between these two opposite requirements. We have looked over existing data structures for representation of switching functions for that where conditions for decomposition of the form (1) are simply expressed. We have noted that EXOR ternary decision diagrams (EXOR-TDDs) [11] are such a data structure for the following reasons.

EXOR-TDDs are introduced for minimization of AND-EXOR expressions [11]. In spectral transforms interpretation of DDs [19], EXOR-TDDs are graphical representations of function expansions in terms of an extended basis consisting of 3^n functions [20]. Thanks to the extension of the basis, the information content of an EXOR-TDD is considerably raised. Thus, in EXOR-TDDs, we sac-

0-7695-0161-3/99 $10.00 © 1999 IEEE

rifice in part compactness, and the reword is highly increased amount of information contained in the EXOR-TDDs. We have pointed out that in EXOR-TDDs, the conditions for decomposability of the form (1) are inherently contained, since they have been already checked during we build EXOR-TDD for f. That permits formulation of a simple procedure to find decomposition of the form (1) through EXOR-TDDs. The same method applies to detection of simple disjoint bi-decomposition [15]. Separation into the necessary and the sufficient conditions, saves many computations in the case when the required decomposition or bi-decomposition is impossible.

2. Matrix-Valued DDs

2.1 Matrix-valued MTBDDs

Multi-Terminal Binary Decision Diagrams (MTBDDs) [6], are a generalization of Binary Decision Diagrams (BDDs) [5] derived by allowing integers or complex numbers as values of constant nodes. Therefore, MTBDDs represents integer or complex-valued functions on finite dyadic groups. Generalization to functions on arbitrary groups is straightforward. In this case, the nodes in MTBDDs are replaced by the nodes with more than two outgoing edges.

Application of non-Abelian algebraic structures in engineering practice [21], implies consideration of matrix-valued functions.

Definition 2: A matrix-valued function is a function defined as a mapping $f : G \rightarrow M_{a,b}$, where G is a finite not necessarily Abelian group, and $M_{a,b}$ is the set of $(a \times b)$ matrices whose entries take values in a field P that may be the complex field C or a finite (Galois) field $GF(p)$.

We denote by $P_{a,b}(G)$ the space of matrix-valued functions on G whose values are $(a \times b)$ matrices with entries in P. To represent functions in $P_{a,b}(G)$, the matrix-valued Decision Diagrams are introduced [18].

Definition 3: The matrix-valued Decision Diagrams (mvDDs) are DDs whose constant nodes take values in $M_{a,b}$.

If G is the finite dyadic group, and $P = C$, then mvDDs in Definition 3 become the matrix-valued MTBDDs (mvMTBDDs). If $M_{a,1}$, and $P = GF(2)$, mvMTBDDs become MTBDD used in [14]. A mvMTBDD is shown in Fig. 2 in Example 1.

2.2 Matrix-valued EXOR-TDDs

Ternary Decision Diagrams (TDDs) are a generalization of BDDs derived by allowing the third outgoing edge for each node [10], [11]. As in BDDs, the first two outgoing edges of a node at the i-th level in TDDs point to the values $f_0 = f(x_i = 0)$, and $f_1 = f(x_i = 1)$. The third edge points to $f_2 = f_0 * f_1$, where $*$ denotes an operation over logic values 0, 1, and u, with u denoting unspecified logic value.

Definition 4: [11] EXOR-TDDs are TDDs where $*$ is the addition in $GF(2)$, usually denoted as logic EXOR.

EXOR-TDDs are derived by the reduction of EXOR-Ternary decision trees (EXOR-TDTs) [11]. In an EXOR-TDT, the constant nodes represent logic values 0 or 1. For a given f, these values are determined a elements of the extended truth-vector $\mathbf{F}_e$ for f. As shown in [20], $\mathbf{F}_e$ is the spectrum for f with respect to the Extended Reed-Muller (ERM) transform.

Definition 5: $\mathbf{F}_e$ for an n-variable switching function $f(x_1, \ldots, x_n)$ given by the truth-vector $\mathbf{F} = [f(0), \ldots, f(2^n - 1)]^T$ is defined by

$$\mathbf{F}_e = \mathbf{E}(n)\mathbf{F}, \quad \mathbf{E}(n) = \bigotimes_{i=1}^{n} \mathbf{E}(1), \quad \mathbf{E}(1) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix},$$

where $\mathbf{E}(n)$ is the ERM-transform matrix. Calculations are performed in $GF(2)$.

From this spectral interpretation, if f is given by a BDD, the EXOR-TDD can be derived by performing calculations determined by $\mathbf{E}(1)$ at each node in the BDD [20]. Similar to the generalization of BDDs into MTBDDs, and further, into mvMTBDDs, we introduce the matrix-valued EXOR-TDDs (mvEXOR-TDDs).

Definition 6: Matrix-valued EXOR-TDDs are a generalization of EXOR-TDDs derived by allowing the matrices over $GF(2)$ as values of constant nodes.

Fig. 3 shows mvEXOR-TDT for f in Example 1, and Fig. 4 shows the corresponding mvEXOR-TDD. The first two outgoing edges in a node are labeled by $\overline{y}_i$ and y_i, respectively. The third edge points to $f_0 \oplus f_1$. Thus, it is denoted by 1, which resembles that $\overline{y}_i \oplus y_i = 1$.

3. Decomposition through mvEXOR-TDDs

The Ashenhurst decomposition theorem [2], can be written in terms of the truth-vector for f as follows.

Theorem 1: For an n-variable switching function f, we split the truth-vector $\mathbf{F} = [f(0), \ldots, f(2^n - 1)]^T$ into 2^r subvectors of 2^{n-r} successive function values. Thus, $\mathbf{F}$ is written as a matrix-valued vector $\mathbf{F}_m = [\mathbf{F}_0, \ldots, \mathbf{F}_{2^{n-r}-1}]^T$. The function f is simple disjoint decomposable with respect to the first r variables iff in $\mathbf{F}_m$ there are at most two different subvectors $\mathbf{F}_i$.

In this case, to each subvector we assign a value $j \in GF(2)$, providing that the same value is assigned to the equal vectors. These values are considered as elements of a vector $\mathbf{H}$ of order 2^r. The vector $\mathbf{H}$ thus generated is the truth-vector for h in (1). The truth-vector of g is $\mathbf{G} = [\mathbf{F}_{i0}, \mathbf{F}_{i1}]^T$, where $\mathbf{F}_{ij}$ is the subvector to which the value $j \in GF(2)$ is assigned.

The same theorem can be written in terms of mvEXOR-TDDs as follows.

We assign to f a matrix-valued function f_m defined in 2^r points. f_m is given by $\mathbf{F}_m$ as defined in Theorem 1. A mvMTBDD for f_m can be easily derived from BDD for f, by simply cutting the last r non-terminal nodes in the BDD. Subfunctions represented by subtrees rooted at the nodes at the $(r-1)$st level in the BDD for f are used as values of constant nodes in the mvMTBDD for f_m.

We derive mvEXOR-TDD for f_m by processing nodes in the mvMTBDD for f_m by the rule described by $\mathbf{E}(1)$, in the same way as we derive EXOR-TDD for f from the

BDD for f. Note that mvEXOR-TDD for f is EXOR-TDD for f_m which is assigned to f.

Let $\mathbf{X}$ and $\mathbf{Y}$ be two arbitrary vectors of order 2^{n-r} over $GF(2)$, and $\mathbf{0}$ the zero-vector of the same order. Let $R = \{\mathbf{X}, \mathbf{Y}, \mathbf{X} \oplus \mathbf{Y}, \mathbf{0}\}$, where $\oplus$ is the componentwise EXOR.

Theorem 2: A switching function f is simple disjoint decomposable with respect to a subset of r variables iff the matrix-valued EXOR-TDD for f_m assigned to f has at most four constant nodes whose values are in R.

Proof. The proof follows from Theorem 1 and definition of EXOR-TDDs. From this theorem, to check decomposability, we have to compare all the subvectors $\mathbf{F}_i$ to see how many different there are. We can do that by the componentwise EXOR of the subvectors $\mathbf{F}_i$. From transitivity, we do not need to form EXOR of all the subvectors. Some EXOR sums are covered by the existence of some others. From definition of EXOR-TDDs, we calculate for the third outgoing edges all the required EXOR sums which are necessary to check the decomposability. Q.E.D.

In mvEXOR-TDD for f, h in (1) is determined by labels at the edges in the paths from the root node to the constant nodes $\mathbf{X}$ and $\mathbf{Y}$. Thus, there are two choices for h. We denote them by h_X and h_Y. The function h_X is determined as the logic sum of labels at the edges in the paths from the root node to the constant node $\mathbf{X}$. h_Y is determined in the same way with respect to the paths to the constant node $\mathbf{Y}$.

The function g in (1) is determined by the values of constant nodes $\mathbf{X}$ and $\mathbf{Y}$, respectively, depending on the chosen h, h_X or h_Y. Thus, g_X corresponding to h_X is given by the truth-vector $\mathbf{G}_X = [\mathbf{X}, \mathbf{Y}]^T$. Similar, the truth-vector for g_Y is $\mathbf{G}_Y = [\mathbf{Y}, \mathbf{X}]^T$.

3.1 Discussion

The simple disjoint decomposability permits at most two different subvectors in $\mathbf{F}_m$. These subvectors produce isomorphic subtrees in the BDD for f.

From definition of EMR-transform, it may be at most four different subvectors in $\mathbf{F}_e$. They are defined as in R in Theorem 2.

By building mvEXOR-TDD for f_m assigned to f, we actually perform comparison of subtrees in the BDD for f. From definition of EXOR-TDDs, comparison is transferred into comparison of constant nodes in the mvEXOR-TDD. Thus, it is done automatically and it is simplified.

Complexity of BDDs, expressed through the number of non-terminal nodes, approximates to $O(2^n/n)$ [16]. Complexity of EXOR-TDDs approximates to $O(3^n/n)$ [11]. From that and the previous considerations, we have the following lemma.

Lemma 1: Complexity of mvEXOR-TDDs for functions which are simple disjoint decomposable with respect to a subset of r variables approximates $O(3^{n-r}/(n-r)+3 \cdot 2^r/r)$, if we use a good ordering of the input variables. If we do not know the order, the complexity is at most $O(3^n/n)$.

Proof. We build EXOR-TDD for f_m which has $(n-r)$ variables. There are three constant nodes, that are $(r \times 1)$ matrices. They can be represented by BDDs, whose complexity is $O(2^r/r)$. The node $\mathbf{0}$, can be replaced by the number-valued node 0. Thus, the Lemma follows. Q.E.D.

For notation convenience, the subvectors $\mathbf{F}_i$ in $\mathbf{F}_m$ may be written as $(p \times q)$ matrices, with appropriately chosen values p and q with $p + q = 2^{n-r}$. In this notation, the p successive values of $\mathbf{F}_i$ are written as a column or row of a $(p \times q)$ matrix.

The condition for decomposability, expressed in terms of mvEXOR-TDDs, can be split into the necessary and sufficient parts. The necessary condition is that mvEXOR-TDD for a simple disjoint decomposable function has at most four constant nodes. The sufficient condition is that the values of these nodes are in R defined in Theorem 2.

For large values of r, matrix-valued constant nodes in mvEXOR-TDD can be represented by BDDs. Fig. 5 shows structure of such a mvEXOR-TDD. Thus, mvEXOR-TDD for f is EXOR-TDD where the constant nodes are BDDs for $\mathbf{F}_i$ in $\mathbf{F}$ [18].

4. Procedure

Theorem 2 permits to formulate a procedure for simple disjoint decomposition with respect to a subset of first r variables in f through mvEXOR-TDDs.

Denote by n_c the number of constant nodes in the mvEXOR-TDD for f_m assigned to f. Denote by c_i, $i = 1, \ldots n_c$ the values of these nodes. The set R is as defined in Theorem 2.

Decomposition Procedure

1. Given an n-variable function f. Assign f_m to f for the required r.
2. Build up mvEXOR-TDD for f_m.
3. Check the number of constant nodes in mvEXOR-TDD from step 2. If $n_c > 4$, the decomposition is impossible. If $n_c \leq 4$, check if $c_i \in R$. If yes, the decomposition is possible.

Example 1: Table I shows a function of five variables, which is used in Examples 11.2.2 and 11.3.1 in [10]. We consider simple disjoint decomposition of this function with respect to the first $r = 3$ variables.

We assign to f the matrix-valued function f_m given by

$$\mathbf{F}_m = \left[\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \right]^T.$$

Fig. 1 shows BDD for f, and Fig. 2 shows mvMTBDD for f. Fig. 3 shows mvEXOR-TDT for f_m. In this figure,

$$\mathbf{X} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \mathbf{Y} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix},$$

$$\mathbf{Z} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \mathbf{0} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}.$$

Fig. 4 shows mvEXOR-TDD for f_m. This mvEXOR-TDD has four constant nodes. Thus, the necessary condition for decomposition of the form (1) is satisfied. If the

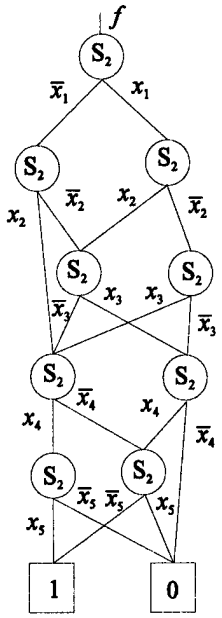


Fig. 1. BDD for f in Example 1.

TABLE I
Function f in Example 1.

		x_3,x_4,x_5							
		0	0	0	0	1	1	1	1
		0	0	1	1	0	0	1	1
		0	1	0	1	0	1	0	1
x_1,x_2	00	1	0	0	1	0	0	1	0
	01	1	0	0	1	1	0	0	1
	10	0	0	1	0	1	0	0	1
	11	1	0	0	1	0	0	1	0

first two constant nodes are denoted by $\mathbf{X}$ and $\mathbf{Y}$ as above, the third node is $\mathbf{X}\oplus\mathbf{Y}$, while the fourth node is $\mathbf{0}$. Thus, f has decomposition of the form (1) with respect to x_1, x_2, x_3, and can be written as $f = g(h(x_1, x_2, x_3), x_4, x_5)$.

We follow the paths from the root node to the constant nodes $\mathbf{X}$ and $\mathbf{Y}$. The products of labels in these paths determine

$$h_X = \overline{y}_1\overline{y}_2\overline{y}_3 + \overline{y}_1 y_2 + y_1\overline{y}_2 y_3,$$
$$h_Y = \overline{y}_1\overline{y}_2 y_3 + y_1 y_2 y_3 + y_1\overline{y}_2\overline{y}_3.$$

The values of constant nodes determine the corresponding g_X and g_Y as follows

$$\mathbf{G}_X = \left[\left[\begin{array}{cc} 1 & 0 \\ 0 & 1 \end{array}\right], \left[\begin{array}{cc} 0 & 1 \\ 0 & 0 \end{array}\right]\right]^T = [1,0,0,1,0,0,1,0]^T,$$

$$\mathbf{G}_Y = \left[\left[\begin{array}{cc} 0 & 1 \\ 0 & 0 \end{array}\right], \left[\begin{array}{cc} 1 & 0 \\ 0 & 1 \end{array}\right]\right]^T = [0,0,1,0,1,0,0,1]^T.$$

5. Bi-Decomposition through mvEXOR-TDDs

Simple disjoint bi-decomposition is another restricted class

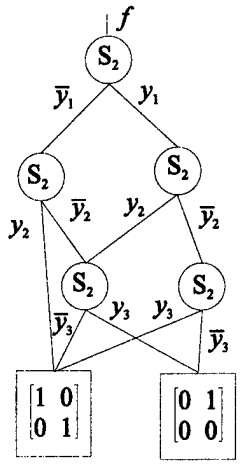


Fig. 2. mvMTBDD for f_m in Example 1.

TABLE II
Function f in Example 2.

		x_3,x_4,x_5							
		0	0	0	0	1	1	1	1
		0	0	1	1	0	0	1	1
		0	1	0	1	0	1	0	1
x_1,x_2,x_3	000	0	0	0	0	0	0	0	0
	001	0	0	0	0	1	1	1	1
	010	0	0	0	0	0	0	0	0
	011	0	0	0	0	1	1	1	1
	100	1	1	1	1	1	1	1	1
	101	1	1	1	1	0	0	0	0
	110	0	0	0	1	0	0	0	1

of functional decomposition [15]. As shown in [15], it is useful in some applications, for example in realizations with programmable devices having two-input circuits at the outputs.

Definition 7: An n-variable switching function f is simple disjoint bi-decomposable iff it can be represented in the form

$$f(x_1,\ldots,x_n) = g(h_1(x_1,\ldots,x_r), h_2(x_{r+1},\ldots,x_n)).$$

Expressed in terms of decomposition charts, f is simple disjoint bi-decomposable iff there are at most two distinct columns and rows, at the same time. Expressed in terms of the truth-vectors, it is required that there are at most two different subvectors of order 2^r in the truth-vectors for $f(x_1,\ldots,x_r,x_{r+1},\ldots,x_n)$ and $f(x_{r+1},\ldots,x_n,x_1,\ldots,x_r)$. In terms of EXOR-TDDs, the conditions for simple disjoint bi-decomposition can be expressed as follows.

Theorem 3: An n-variable switching function f is simple disjoint bi-decomposable, iff the set R in Theorem 2 is $R = \{\mathbf{X}, \mathbf{Y}, \mathbf{1}, \mathbf{0}\}$, where $\mathbf{1}$ and $\mathbf{0}$ are the constant matrices whose all the elements are equal to 1 and 0, respectively.

Proof: The proof directly follows from Theorem 2 and definition of simple disjoint bi-decomposition for f.

Example 2: Consider a function f given in Table II, used in Example 5.1 in [15]. Assume that we want to check EXOR bi-decomposability with respect to the subsets of variables

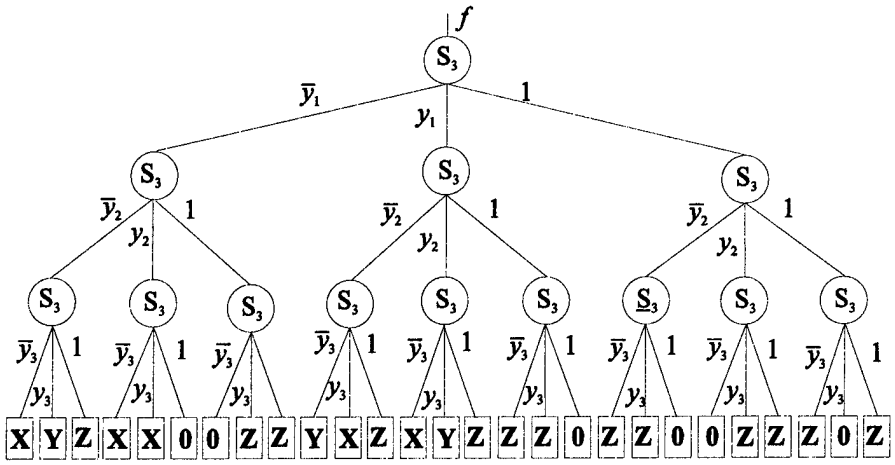


Fig. 3. mvEXOR-TDT for f_m in Example 1.

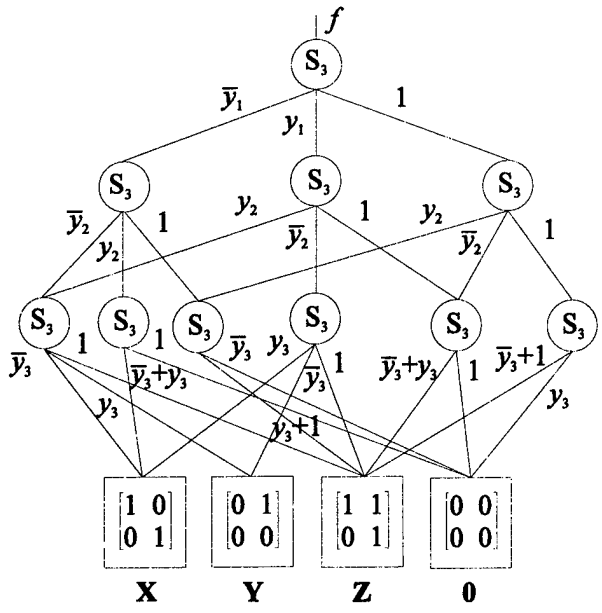


Fig. 4. mvEXOR-TDD for f in Example 1.

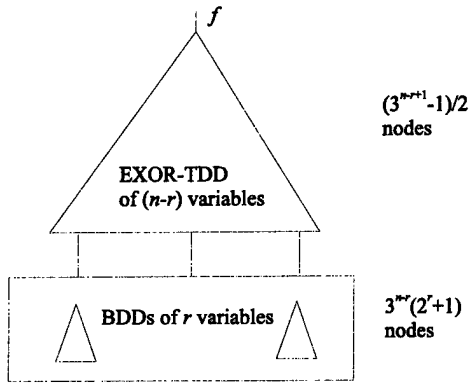


Fig. 5. Structure of mvEXOR-TDTs with BDTs as constant nodes.

$X_1 = \{x_1, x_2, x_5, x_6\}$ and $X_2 = \{x_3, x_4\}$. Thus, we want to represent f as $f = g_1(X_1) \oplus g_2(X_2)$. To make the example obvious, we write f as a positive polarity Reed-Muller (PPRM) expression $f = x_1 \oplus x_1x_2 \oplus x_3x_4 \oplus x_1x_2x_5x_6$.

The method derived from Theorem 3 is formulated for decomposition with respect to the first r variables in f. Therefore, we determine an auxiliary function $f'(x_3, x_4, x_1, x_2, x_5, x_6) = f(x_1, x_2, x_3, x_4, x_5, x_6)$. Thus, $f' = x_3 \oplus x_3x_4 \oplus x_1x_2 \oplus x_3x_4x_5x_6$. We associate to f' a matrix-valued function f'_m. Elements of f'_m are matrices of four successive values of f'. Thus, f'_m is given by the matrix-valued truth-vector $\mathbf{F}'_m = [\mathbf{0}, \mathbf{0}, \mathbf{1}, \mathbf{X}, \mathbf{0}, \mathbf{0}, \mathbf{1}, \mathbf{X}, \mathbf{0}, \mathbf{0}, \mathbf{1}, \mathbf{X}, \mathbf{1}, \mathbf{1}, \mathbf{0}, \mathbf{Y}]^T$, where $\mathbf{X} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$, and $\mathbf{Y} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$.

To check the required bi-decomposability, we should determine how many different subvectors there are in $\mathbf{F}'_m$. In that order, we should mutually compare all the elements in $\mathbf{F}'_m$. In simple examples, as that we are considering, this is obvious. However, in functions with many variables, that is a time consuming task, since the number of comparisons is large. In the method we are proposing, we perform these comparisons by building mvEXOR-TDD for $\mathbf{F}'_m$.

Fig. 6 shows mvEXOR-TDD for f'_m. The constant nodes in this mvEXOR-TDD may be denoted as $\mathbf{X}$, $\mathbf{Y}$, $\mathbf{1}$ and $\mathbf{0}$, respectively. Thus, they belong to R as defined in Theorem 3, and simple disjoint bi-decomposition of f' with respect to the subsets of variables x_1, x_2 and x_3, x_4, x_5, x_6 is possible. It follows that the same decomposition for f is possible with respect to the subsets of variables X_1 and X_2. Thus, f can be written as $f = g_1(X_1) \oplus g_2(X_2)$.

If the node $\mathbf{X}$ in the mvEXOR-TDD in Fig. 6 is taken as the truth-vector for g_2, then $\mathbf{G}_2 = [0, 0, 0, 1]^T$. This is a two-variable function $g_2 = z_1z_2$, or in terms of variables in X_2, $g_2 = x_3x_4$. We determine g_1 in the required decomposition by following the paths from the root node to the node $\mathbf{X}$ in the mvEXOR-TDD for f'_m. From Fig. 6 we determine

$$g'_1 = \overline{y}_1\overline{y}_2y_3y_4 \oplus \overline{y}_1\overline{y}_2y_4 \oplus \overline{y}_1y_2y_3y_4 \oplus \overline{y}_1y_2y_4 \oplus y_1\overline{y}_2y_3y_4 \oplus y_1\overline{y}_2y_4 \oplus y_1y_2y_4.$$

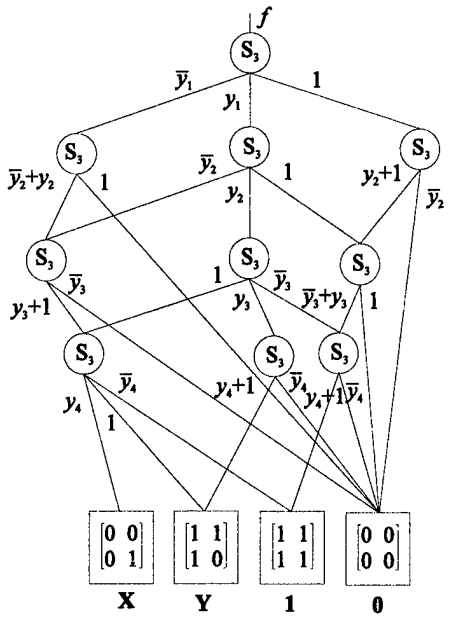


Fig. 6. mvEXOR-TDD for f'_m in Example 2.

In terms of the variables in X_1,

$$g'_1 = \overline{x}_1\overline{x}_2x_5x_6 \oplus \overline{x}_1\overline{x}_2x_6 \oplus \overline{x}_1x_2x_5x_6 \oplus \overline{x}_1x_2x_6 \oplus x_1\overline{x}_2x_5x_6 \oplus x_1\overline{x}_2x_6 \oplus x_1x_2x_6.$$

We derive g_1 in bi-decomposition for f by performing on g'_1 the permutation inverse to that used to determine f' from f. We pay attention that the group of permutations is a non-Abelian group, which determines the order of permutations we should perform. It follows that, in this example, we should permute $x_1 \leftrightarrow x_6$ and $x_2 \leftrightarrow x_5$. Thus,

$$g_1 = \overline{x}_6\overline{x}_5x_2x_1 \oplus \overline{x}_6\overline{x}_5x_1 \oplus \overline{x}_6x_5x_2x_1 \oplus \overline{x}_6x_5x_1 \oplus x_6\overline{x}_5x_2x_1 \oplus x_6\overline{x}_5x_1 \oplus x_6x_5x_1.$$

After reduction, we get $g_1 = x_1 \oplus x_1x_2 \oplus x_1x_2x_5x_6$, which is obvious from PPRM for f. The same result is derived in [15] by the analysis of PPRM for f.

6. Closing Remarks

We considered simple disjoint decomposition and simple disjoint bi-decomposition of switching functions through EXOR-TDDs. It is shown that to check decomposability or bi-decomposability with respect to r variables is equivalent to build EXOR-TDD for a matrix-valued function of $(n-r)$ variables. In that order, the matrix-valued EXOR-TDDs are introduced. In practical realizations, these TDDs are represented as EXOR-TDDs with BDDs as constant nodes. In practice, we do not always look for the fastest methods, using possibly quite specialized tools. We may not always look for methods and tools that can handle the most complex problems. Sometimes, we are quite satisfied if we can solve a few tasks with a single tool that we have. For that case, we recommend EXOR-TDDs.

Decomposition and bi-decomposition are two more tasks, which, together with several other tasks in logic design, can be efficiently solved by EXOR-TDDs. The method is applicable to functions of a moderate complexity.

Acknowledgment

The author is very grateful to Prof. Tsutomu Sasao for many valuable comments and suggestions. Thanks are due to the referees whose comments improved presentation in this paper.

References

[1] S.B. Akers, "Binary decision diagrams" *IEEE Trans. on Computers*, Vol. C-27, No. 6, 1978, 509-516.

[2] R.L. Ashenhurst, "The decomposition of switching functions", *Proc. Int. Symp. on the Theory of Switching*, April 1957, 74-116.

[3] R.I. Bahar, E.A. Frohm, C.M. Gaona, G.D. Hachtel, E. Macii, A. Pardo, and F. Somenzi, "Algebraic decision diagrams and their applications," *ICCAD*, November 1993, 188-191.

[4] V. Bartacco, M. Damiani, "The disjunctive decomposition of logic functions", *ICCAD'97*.

[5] R.E. Bryant, "Graph-based algorithms for Boolean functions manipulation," *IEEE Trans. Comput.*, Vol.C-35, No.8, 1986, 667-691.

[6] E.M. Clarke, K.L. McMillan, X. Zhao, M. Fujita, J. Yang, "Spectral transforms for large Boolean functions with application to technology mapping," *DAC-1993*, June 1993.

[7] Y.-Y. Lai, M. Pedram, S.B.K. Vrudhula, "EVBDD based algorithms for integer programming, spectral transformation, and functional decompositions", *IEEE Trans. on CAD*, Vol. 13, No. 8, 1994, 959-975.

[8] Y. Matsunaga, "An attempt to factor logic functions using exclusive-or decomposition", *SASIMI'98*, 78-83.

[9] S. Minato, G. De Micheli, "Finding simple disjunctive decompositions using irredundant sum of products forms" *ICCAD'98].*

[10] Sasao, T., "FPGA design by generalized functional decomposition", in: T. Sasao, (ed.), *Logic Synthesis and Optimization*, Kluwer Academic Publishers, 1993, 233-258.

[11] T. Sasao, "Ternary decision diagrams and their applications," *ISMVL-97*, May 1997, 241-250.

[12] T. Sasao, *Logic Design: Switching Circuit Theory*, Second edition, Kindai Kagakusha, Tokyo, Japan, 1998.

[13] T. Sasao, "On a method to accelerate functional decompositions", *Tech. Rept. of IEICE*, VLD98-58, ICD98-161, FTS98-85 (1998-09), 103-109.

[14] T. Sasao, J.T. Butler, "A method to represent multiple-output functions by using multi-valued decision diagrams," *ISMVL-26*, May 1996, 248-254.

[15] T. Sasao, J.T. Butler, "On bi-decompositions of logic functions", *Proc. Int. Workshop on Logic Synthesis*, Lake Tahoe, California, May 18-21, 1997.

[16] T. Sasao, M. Fujita, (ed.), *Representations of Discrete Functions*, Kluwer Academic Publishers, 1996.

[17] T. Sasao, M. Matsuura, "DECOMPOS: An integrated system for functional decomposition", *1998 Int. Workshop on Logic Design*, Lake Tahoe, June 1998. Also in: *Tech. Rept. of IEICE*, VLD98-57, ICD98-160, FTS98-84, (1998-09), 95-102.

[18] R.S. Stanković, "Fourier decision diagrams on finite non-Abelian groups with preprocessing," *ISMVL-27*, May 1997, 281-286.

[19] R.S. Stanković, T. Sasao, C. Moraga, "Spectral transform decision diagrams," in: [16], 55-92.

[20] R.S. Stanković, T. Sasao, "Spectral Interpretation of TDDs," *SASIMI'97*, 45-50.

[21] E.A. Trachtenberg, "Applications of Fourier Analysis on Groups in Engineering Practices", in: Stanković, R.S., Stojić, M.R., Stanković, M.S., (eds.), *Recent Developments in Abstract Harmonic Analysis with Applications in Signal Processing*, Nauka, Belgrade and Elektronski fakultet, Niš, 1996, 331-403.

Synthesis of Multiple-Valued Decision Diagrams using Current-Mode CMOS Circuits

Mostafa Abd-El-Barr
Department of Computer Engineering
King Fahd University of Petroleum & Minerals
Dhahran 31261
Saudi Arabia
mostafa@ccse.kfupm.edu.sa

Henry Fernandes
Department of Computer Science
University of Saskatchewan
Saskatoon S7N 0W0
Canada
hff135@cs.usask.ca

Abstract

In this paper, an algorithm for generating modular designs of Ordered Multiple Decision Diagrams (OMDDs) for Current-Mode CMOS Logic (CMCL) implementation is introduced. The OMDD structures for a set of twelve benchmark circuits from the LGSynth93 using radices ranging from r =2 to r =10 are generated and compared in terms of size and speed. It is observed that MODDs with radices r ∈ {2, 4, 8} result in the smallest area. They also achieve the smallest normalized (with respect to the AT^2 measure for r = 2) AT^2, where A is the area and T is the delay.

1 Introduction

Multiple Decision Diagrams (MDDs) are graph structures which are becoming the state-of-the-art means for representation of both binary and multiple-valued logic functions [1][2]. A number of research studies on MDDs appeared recently in the literature. These include [3][4][5][6][7][8]. In [3], some of the issues concerning implementation of an MDD package are considered. In particular, the issue of adjacent level interchange to allow for logical operations to be performed on reduced OMDDS (ROMDDs) has been studied. A preliminary package has been implemented in C. In [4] new techniques, e.g. inverted edges and cofactoring have been employed in the synthesis of MVL MDDs. The integration of the proposed techniques into CAD environment has also been made. In [5] a design method for mutliple-output networks using time divison multiplexing and shared multi-terminal MVL MDDs has been introduced. It is shown that pairing of input variables and pairing of output functions can lead to a substantial reduction in the number of gates needed for the implementation of functions. A comprehensive coverage of Ternary Decision Diagrams (TDDs) can be found in [6].

In this paper, we consider ordered MDDs (OMDDs) for representation and minimization of binary functions using Current-mode CMOS (CMCL) MVL circuits [9][10][11]. We have generated complete layout for OMDD basic cells using CMOS3DLM design rules. Several benchmark circuits have been selected from the LGSynth93 [12] and their OMDD structures are generated for varying radices. HSPICE level3 modeling and CMOS3DLM design rules are used in estimating the area and the delay of the generated OMDD structures for the benchmark circuits considered [9].

2 Background Material

We consider a given r-valued function $f(X)$, where $X = \{x_1, x_2, \ldots, x_n\}$ as a mapping f: $R^n \rightarrow \{0, 1\}$, where $R = \{0, 1, 2, \ldots, r\text{-}1\}$. When writing a product term of a function, the value of each variable is presented superscripted. For example, a product term in a 4-variable 4-valued function can be written as $p = A^2B^1C^0D^3$. The meaning of this representation is that the product term, p, = 1 if A =2 & B = 1 & C = 0 & D = 3. The same product term can be written as p(A, B, C, D) = 2103_4. A representation of a an n-variable $r-$valued function, f, with k product terms will be given as $f = \sum(p_1, p_2, \ldots, p_k)$. For example, a 4-variable 4-valued function will be represented as $f = \sum(2103_4, 2321_4, 1110_4)$.

A MDD is acyclic graph (DAG) which forms a canonical representation of a given function. An example of

0-7695-0161-3/99 $10.00 © 1999 IEEE

an MDD is shown in Figure 1 for the function $f = \sum(101_3, 110_3, 111_3, 112_3) = A^1B^0C^1 + A^1B^1$. In order

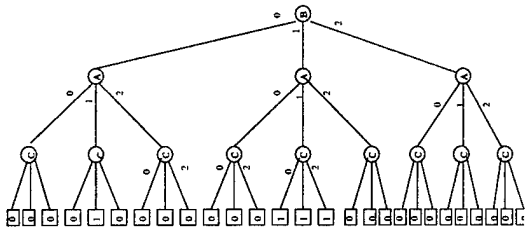


Figure 1. Example of a 3-variable 3-valued MDD representation of a binary function.

to make MDDs more useful structures, some restrictions can be imposed. An OMDD is an MDD with some ordering $x_i < x_j < \cdots < x_k$ imposed upon the variables. Figure 2 shows the effect of variable ordering for the binary function $F = AB + CD + EG$. In this figure, a solid line indicates a "0" value for the variable while a dashed line indicates a "1" value for the variable. As can be seen from the figure, the number of nodes has been reduced from 14 to 6 when the variable ordering A, B, C, D, E, G is used instead of the ordering G, C, A, E, B, D. To further reduce the OMDDs in size, the following two rules can be imposed.

1. Remove duplicate nodes , and
2. Remove redundant nodes (a redudant node has all its output paths leading to the same node).

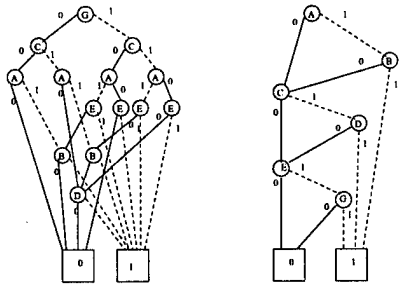


Figure 2. The effect of variable ordering on the size of a MDD.

Fig. 3 illustrates the effect of applying the above rules on the size of the MDD shown in Fig. 1. Fig. 4 illustrates the basic CMCL MVL circuits used. These circuits are respectively the current generator, current mirror, threshold detector, and the switch. Detailed operation of these cells can be found in [10].

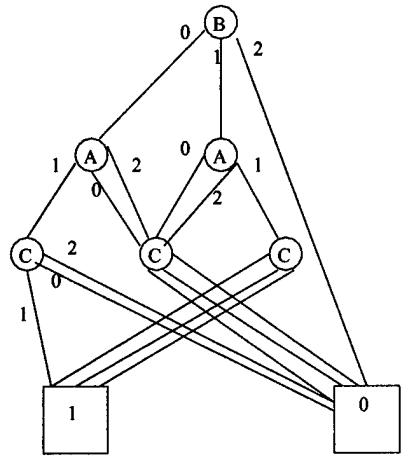


Figure 3. The effect of removing duplicate nodes on the size of an OMMS.

3 OMDDs Implementation

The OMDD consists of a number of identical nodes (cells). A schematic of a general node is shown in Fig. 5. The truth table of the cell is shown in Table 1. The CMCL MVL implementation of a node for r=4 is shown in Fig. 6. The cell has one input and 3 (r-1) outputs. It should be noted that the threshold element, *Th(0)*, is not connected to the P-*mirror*. This is because the value of the input current I_{in} is assumed to be $I_{in} \geq 0$. This necessitates the modification of the *threshold* element shown in Fig. 4 to become as shown in Fig. 7. This has been done in order to guarantee that when $V_{in} = 0$, then the output of *Th(0)* will be 0 due to pre-discharging of the node y using the signal CLK. The cell operates as follows. When V_{in} is binary high and according to the value of the input current I_{in}, the output of one of the threshold elements will be binary high and the outputs of the remaining threshold elements will be binary low. This binary high value will propagate through the transmission gates to the inverter input where it will be inverted. The inverted value will then enable the corresponding output p transistor, thus producing a pure logic one at the corresponding output O_i. It should be noted that the output of the node are voltage levels. This allows us to derive multiple subsequent (multiple fan-out) nodes, thus allowing the implementation of multiple functions with a single OMDD.

4 OMDD Generation

In this section, we introduce the algorithm used to generate the OMDD for a given function using the gen-

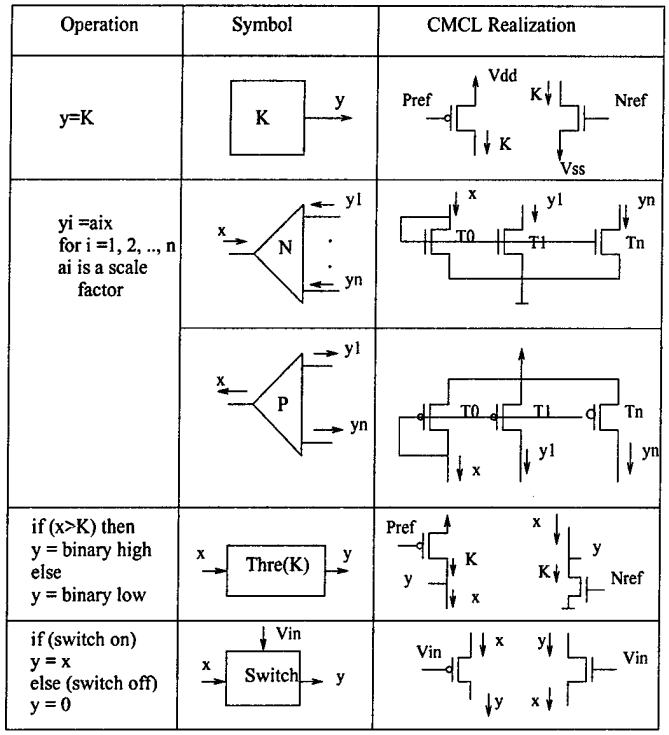


Figure 4. CMCL MVL basic cells.

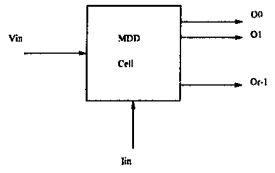


Figure 5. Schematic of a cell (node) in an OMDD.

Table 1. Truth table of the OMDD basic cell.

V_{in}	I_{in}	O_0	O_1	...	O_{r-1}
0	x	x	x	...	x
1	$0I_0$	1	0	...	0
1	$1\ I_0$	0	1	...	0
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
1	$(r\text{-}1)I_0$	0	0	...	1

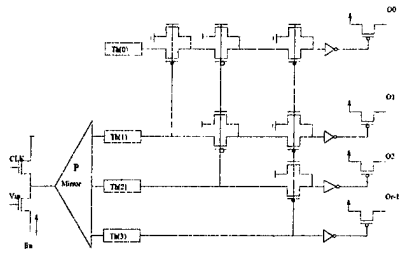


Figure 6. CMCL MVL implementation of the node in an OMDD for r = 4.

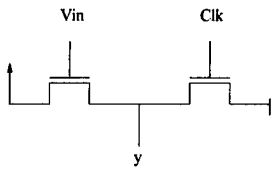


Figure 7. The modified threshold element used for the Th(0) in Fig. 4.

eral node structure shown in Fig. 5 as the basic cell.

The Algorithm

In the following algorithm, the variables used have the following meanings.
numVars: the number of variables in the function
numF: the number of functions
node: the number assigned to the node currently generated
max: the maximum assigned for nodes generated so far
$L_{i,j}$: the set of minterms of function j that may still be satisfied at node i
$level_i$: the level number of node i
(val, num): returns the node number reached from node *val* along path *num*

1. node ← 2
2. $level_2$ ← 1
3. L_2 ← set of all minterms of F
4. max ← 2
5. for i ← 1 to r-1
   a) Let L_{max+1} be the list of all minterms of L_{node} with the $level_{node}^{th}$ variable = i.
   b) if $|L_{max+1}| = 0$ then (node,i) ← 0.
   c) if $|L_{max+1}| = r^{numVars+1-level_{node}}$ then (node,i) ← 1
   d) if L_{node} is equal to L_j (where j < node), then (node,i) ← (j,i).
   e) if (node,i) hasn't been assigned yet, then max ← max + 1
      (node,i) ← max
      $level_{max}$ ← $level_{node}$ + 1.
6. if max > node then node ← node + 1 and goto 5.
7. for node ← max to 2
   if (node,i) = (node,0) ∀ i≤r-1 then for j ← 2 to node-1
   if (j,k) = node then (j,k) = (node,0), ∀ k ≤ r-1

The algorithm used for a given variable ordering iteratively cycles through each node in the OMDD and for each path steming from that node, the algorithm determines the subsequent node to which this path leads. A value of 1(0) is assigned in order to show that the value of the function is 1(0) if such path is followed. The subsequent node to which a path should lead is determined as follows. For each path, a list of possible

minterms which can still be covered by following that path is generated. Three possibilities exist. First, if the list is empty, then the path leads to the value 0 for the function. Second, if the path covers all possible remaining mintermins for the given function, then that path leads to the value 1 for the function. The third possibility is when neither of the above two cases is true. In this cases, the path is assumed to lead to an intermediate node in the OMDD and a new node is generated unless an identical node exists in the OMDD (no duplication nodes are allowed). If such node exists, then the algorithm assigns that node to the path. The algorithm continues in this fashion until leaf nodes are reached. At that time, the algorithm cycles backwards through all nodes trying to identify redundant nodes and remove them. The algorithm used is shown below.

4.1 An Example

Consider a 3-variable 3-valued function given by $F = A^0B^1C^2 + A^1B^0C^1 + A^1B^1C^1 + A^1B^1C^2$. $= \sum(012_3,\ 101_3,111_3,\ 112_3)$. For this function, the variable ordering A, C, B is assumed.

Output		Level	Set of Minterms
2	3 4 0	1	(012, 101, 111, 112)
3	0 0 5	2	(-12)
4	0 6 5	2	(-01, -11, -12)
5	0 1 0	3	(-1-)
6	1 1 0	3	(-0-, -1-)

The program starts with node 2. The reason for this is that the numbers 1 and 0 are reserved for terminal nodes. Using L_2 = (012, 101, 111, 112), we determine the output paths of this node. With the restriction A = 0, we get L_3 = (-12). The '-' is just used here to signify that the variable A has already been covered. Next, we use the restriction A = 1. This gives us L_4 = (-01, -11, -12). After this, we try A = 2. Since this would give us an empty list, this value would be 0. Thus, we get (1,0) = 3, (1,1) = 2, and (1,2) = 0.
NOTE : The addition of the dashes here, although not mentioned in the above algorithm, is necessary. The dashes show how two minterms can share a node. For example, take the minterms 012 and 112. After the A restriction, they become -12 and -12. Thus, they now appear identical and can be both covered by the same nodes in the OMDD.

5 Experimental Results

We have applied the above algorithm to twelve benchmark circuits from the LGSynth93 [12]. In each of these circuits, the OMDD strcutures has been generated for radices ranging from r =2 to r = 10. The OMDD delay and physical dimension of the OMDDs are estimated. The delay has been obtained using HSPICE CMOS3DLM parameters, while the area measurements were made using the 1.5 micron, two metal process in *Cadence*. More elaboration on these two aspects is given below.

5.1 Delay Characteristics

Fig. 8 shows the OMDD structure generated for the "9symml" benchmark circuit for the case r = 2. There are 33 nodes in total in that OMDD. Solid lines in this figure indicate paths to "0" value while dashed lines represent paths to "1" value. We have estimated the delay for that OMDD as $T_{OMDD} = N_{variables} \times T_{node}$, where $N_{variables}$ represents the number of variables in the function and T_{node} represents the time delay of a single node. Table 2 shows the average delay of a node for each of the nine radices r =2 to r = 10. These node delay values are obtained using HSPICE.

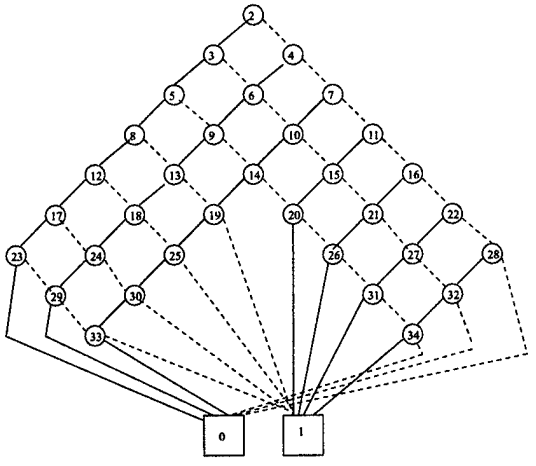


Figure 8. The OMDD structure for the "9symml" benchmark circuit for r =2.

Table 2. The average node delay in "9symml" for different radices

Radix	Average delay (ns)
2	5.1
3	8.4
4	11.6
5	15.5
6	19.7
7	23.8
8	28.8
9	34.0
10	39.9

5.2 Area Estimation

We have enumerated the number of nodes in the OMDD of twelve benchmark circuits for radices ranging from $r = 2$ to $r = 10$. Table 4 lists the the benchmark circuits and the number of nodes in their OMDD for the radices considered.

Table 3. The number of nodes in the OMDD of a number of benchmark circuits.

Circuit	r=2	r=3	r=4	r=5	r=6	r=7	r=8	r=9	r=10
9symml	33	62	17	40	37	33	11	28	26
C17	12	15	8	9	6	6	5	5	5
Z5xp1	127	66	43	35	27	23	19	18	16
a2	11	8	5	5	4	4	3	3	3
apex4	438	233	153	117	95	80	66	59	54
con1	28	43	14	32	23	23	9	17	16
dc1	11	7	4	3	3	3	3	3	1
fsm	16	16	9	10	7	6	5	5	5
majority	9	11	5	9	6	8	4	5	5
mlp4	243	127	80	64	52	42	35	33	29
rd84	36	79	16	56	35	37	10	33	29
xor5	9	11	5	9	5	6	3	5	5

From table 4, one can see that the number of nodes in the realization of a given OMDD is minimal for r = 2, 4, and 8. In order to estimate the actual silicon area consumed by a given OMDD, we have investigated three ways to estimate such area. These are:

1. Taking interconnect into account: Here the height of the OMDD at its highest point is multiplied by the length of the OMDD. Added to this is the interconnect area.
2. Only node area is considered: Here the structure of the OMDD is ignored. We just mulitply the number of nodes times the area consumed by the node for a given radix (see table 4).
3. Taking the area of nodes and the interconnect among nodes into account: Here we take the area consumed by interconnects among nodes into account.

Fig. 9 illustrates the difference between the area measurments as given by points 2 and 3 above.

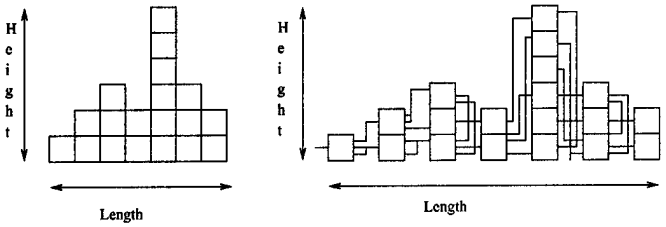


Figure 9. Two different area measurements.

Table 4. The area of the OMDD cells for different radices.

Radix	Cell Area (μm^2)
2	3480
3	5371
4	8215
5	11765
6	15580
7	20240
8	23900
9	31360
10	37820

We have used a combined measure in order to assess the merits of the generated OMDDs. This measure is the AT^2 figure of merits, where A is the area and T is the delay. In order to compute such figure of merits, we computed the product of the number of nodes in the OMDD (as shown in Table 3) times the area of one node (as shown in Table 4) times the square of the time delay of each node (as shown in Table 2). We then normalized such factor with respect to the AT^2 for the case $r = 2$. The resultant normalized figures are shown in Table 5. It is clear from table 5 that OMDDs with radices r=2, 4, and 8 possess the best figure of merits measured by the factor AT^2.

6 Conclusion

In this paper, we have considered the issues related to synthesis of OMDDs for current-mode CMOS MVL implementation. The basic cell of the OMDD has been designed and its performance has been evaluated in terms of the chip area and speed at different radices ranging from r=2 to r = 10. An algorithm has been proposed for the generation of OMDDs using the design

Radix	Normalized AT^2
2	1
3	8
4	6
5	38
6	75
7	127
8	73
9	340
10	524

Table 5. The AT^2 measures for the "9symml" benchmark circuit for different radices

cell. The OMDDs for a number of benchmark circuits from the LGSynth93 have been generated and their performance has been compared in terms of the chipe area, the delay, and the AT^2 figure of merits. It has been shown that radices r =2, 4, and 8 lead to OMDDs with the smallest area and the best AT^2. An important contribution of this work is laying the foundation for a CAD tool for synthesis of OMDDS using CMCL MVL basic cells.

Acknowledgment

The authors wowld like to acknowledge KFUPM for its continued support.

References

[1] R. Bryant. Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams. *ACM Computing Surveys*, 24(3):293–318, September 1992.

[2] A. Lloris, J. Gomez, and R. Roman. Using decision trees for minimization of multiple-valued functions. *International Journal of Electronics*, 75(6):1–35–1041, June 1993.

[3] D. Miller and R. Drechsler. Implementing a Multple-Valued Decision Diagram Package. In *Proc. 28th ISMVL*, pages 52–57, Fukuoka, Japan, May 1998.

[4] L. Macchiarulo and P. Civera. Ternary Decision Diagrams with Inverted Edges and Cofactoring - An Application to Discrete Neural Networks Synthesis. In *Proc. 28th ISMVL*, pages 58–64, Fukuoka, Japan, May 1998.

[5] H. Babu and T. Sasao. Design of Multiple-output Networks using Time Domain Multiplexing and Shared Multi-Terminal Multiple-Valued Decision Diagrams. In *Proc. 28th ISMVL*, pages 45–51, Fukuoka, Japan, May 1998.

[6] T. Sasao. Ternary Decision Diagrams - A Survey. In *Proc. 27th ISMVL*, pages 241–250, Nova Scotia, Canada, May 1997.

[7] D. Miller and N. Muranaka. Multiple-Valued Decision Diagrams with Symmetric Variable Nodes. In *Proc. 26th ISMVL*, pages 242–247, Santiago de Compostela, Spain, May 1996.

[8] T. Sasao and J. Butler. A Method to Represent Multiple-Output Switching Functions by Using Mault-Valued Decision Diagrams. In *Proc. 26th ISMVL*, pages 248–254, Santiago de Compostela, Spain, May 1996.

[9] A. Jain. Multiple-Valued Logic Design in Current-Mode CMOS. In *PhD Thesis, University of Saskatchewan*, Saskatoon, Canada, June 1994.

[10] A. Jain, R. Bolton, and M. Abd-El-Barr. CMOS Multiplei-Valued Logic Part I: Circuits Implementation. *IEEE Transactions on Circuits and Systems*, 40(8):503–514, August 1993.

[11] Y. Chang and J. Butler. The design of Current Mode CMOS Multiple-Valued Circuits. In *Proc. 21st ISMVL*, pages 130–138, May 1991.

[12] K. McElvain. Logic Synthesis and Optimization Benchmarks. In *Proc. 1993 International Logic Synthesis Workshop*, May 1993.

Shared Multiple-Valued Decision Diagrams for Multiple-Output Functions

Hafiz Md. Hasan Babu and Tsutomu Sasao
Department of Computer Science and Electronics
Kyushu Institute of Technology
Iizuka 820, Japan

Abstract

In this paper, we propose a method to represent multiple-output functions using shared multiple-valued decision diagrams (SMDDs). We show an algorithm for pairing the input variables of binary decision diagrams (BDDs). We also present the pair sifting that moves pairs of 4-valued input variables to speed up the normal sifting, and to produce compact SMDDs. The size of the SMDD is the total number of non-terminal nodes excluding the nodes for output selection variables. We derive the sizes of SMDDs for general functions and symmetric functions. From experimental results, we conjecture that, for $n > 1$, the sizes of SMDDs for bit-counting functions (wgt n) and incrementing functions (inc n) are $n\lfloor \log_2 n \rfloor + n - 2^{\lfloor \log_2 n \rfloor}$ and $2n - 1$, respectively, where n is the number of binary input variables, and $\lfloor a \rfloor$ denotes the largest integer not greater than a. We also compare our method with other one. Experimental results show the efficiency of our method.

1 Introduction

Multiple-valued decision diagrams (MDDs) are extensions of binary decision diagrams (BDDs), and are useful in logic synthesis, time-division multiplexing (TDM) realizations, logic simulation, FPGA design, etc [3, 5, 8, 9, 11, 12, 13, 14]. MDDs are usually smaller than the corresponding BDDs, and require fewer memory access to evaluate them [2, 15]. A shared multiple-valued decision diagram (SMDD) is a set of MDDs that compactly represents a multiple-output function [2, 6]. SMDDs can be used in many applications such as design of multiplexer-based networks, design of pass-transistor logic networks, etc [9, 13, 14]. For example, Fig. 1.2 shows the multiplexer-based network corresponding to the SMDD in Fig. 1.1. In these applications, the reduction of the number of nodes in the SMDDs is important. In [2], Sasao and Butler used shared binary decision diagrams (SBDDs) to find good pairs of the input variables. In [6], Miller and Drechsler used MDDs to represent functions with 4-valued outputs, and minimized them by cycle negations. This paper considers the following methods to construct compact SMDDs:

1. Pair the binary input variables to make multiple-valued variables.
2. Order the multiple-valued variables in the SMDDs.

We will introduce a parameter to find good pairs of the input variables. The parameter of an input variable denotes the influence of the variable on the size of the BDD. We also present an extension to the sifting algorithm that moves pairs of 4-valued input variables to speed up the sifting, and to produce compact SMDDs. Furthermore, from experimental results, we derive formulas for the sizes of SMDDs for bit-counting functions (wgt n) and incrementing functions (inc n). The rest of the paper is organized as follows: Section 2 defines various decision diagrams, and shows the properties of SMDDs. Section 3 presents an algorithm for pairing the input variables, and shows optimization methods for SMDDs. Section 4 presents experimental results for various benchmark functions. It shows the efficiency of our method, and compares the sizes of SMDDs obtained by two different methods.

2 Decision Diagrams

In this section, we define various decision diagrams, and present properties of shared multiple-valued decision diagrams (SMDDs). The proofs of lemmas and theorems are omitted due to the space limitation.

Definition 2.1: *Let $F = (f_0, f_1, \ldots, f_{m-1})$. **The size of a decision diagram (DD) for a function F,** denoted by size(DD, F), is the total number of non-terminal nodes excluding the nodes for output selection variables.*

Example 2.1: *The size of the SMDD in Fig. 1.1 is 7. Note that g_0 is the output selection variable in the SMDD.* ■

0-7695-0161-3/99 $10.00 © 1999 IEEE

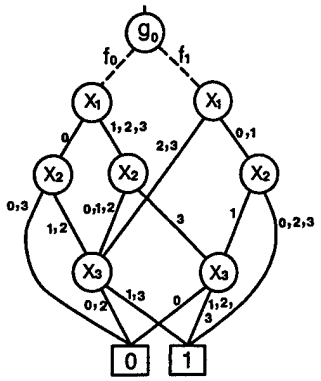


Figure 1.1: SMDD.

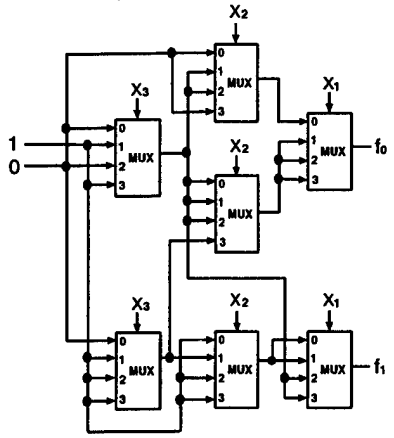


Figure 1.2: A multiplexer-based network corresponding to the SMDD in Fig. 1.1.

2.1 Binary Decision Diagrams

Binary decision diagrams (BDDs) are efficient representations of logic functions [1]. A shared binary decision diagram (SBDD) is a set of BDDs combined by a tree for output selection [4, 7], and represents a multiple-output function.

2.2 Multiple-Valued Decision Diagrams

Let $f : \{0,1,\ldots,r-1\}^N \rightarrow \{0,1\}$. A multiple-valued decision diagram (MDD) of an r-valued N input variables function $f(X_1,X_2,\ldots,X_N)$ is a directed graph with a root node that has r outgoing edges labeled $0,1,\ldots,r-1$ directed to nodes representing $f(0,X_2,\ldots,X_N)$, $f(1,X_2,\ldots,X_N)$, and $f(r-1,X_2,\ldots,X_N)$, respectively. For each of these nodes, there are r outgoing edges which go to nodes that have r outgoing edges, etc. A terminal node is a node that has no outgoing edges. It is labeled by 0 or 1 which corresponds to a binary value of the function f. A reduced ordered MDD (ROMDD) is derived from a multiple-valued complete decision tree using the following reduction rules:

- Two nodes are merged into one node if they represent the same function, and
- A node v is deleted if all the children of v represent the same function.

From now, we will simply call an ROMDD as an MDD.

2.2.1 Shared Multiple-Valued Decision Diagrams

Shared multiple-valued decision diagrams (SMDDs) represent multiple-valued multiple-output functions. An SMDD is a set of MDDs combined by a tree for output selection. Fig. 1.1 shows an example of an SMDD, where g_0 is the output selection variables for functions f_0 and f_1. An SMDD has the following properties:

1. It easily checks the equivalence of two functions.
2. It shares isomorphic sub-graphs of MDDs, and represents a multiple-output function compactly.
3. Its logic levels are smaller than BDDs.

Theorem 2.1: *Let* $R = \{0,1,\ldots,r-1\}$ *and* $B = \{0,1\}$. *Then, the size of the SMDD for an* N*-input* m*-output function* $R^N \rightarrow B^m$ *is at most* $\min_{k=1}^{N} \{m \cdot \frac{r^{N-k}-1}{r-1} + 2^{r^k} - 2\}$.

Corollary 2.1: *An arbitrary* N*-input* m*-output function* $R^N \rightarrow B^m$ *can be represented by an SMDD with* $O(\frac{m \cdot r^N}{N})$ *nodes.*

Lemma 2.1: *Let* $R = \{0,1,\ldots,r-1\}$ *and* $B = \{0,1\}$. *Then, all the non-constant symmetric functions* $R^N \rightarrow B$ *can be represented by MDDs with* $\sum_{i=1}^{N} [2^{\binom{i+r-1}{i}} - 2]$ *non-terminal nodes.*

Theorem 2.2: *Let* $R = \{0,1,\ldots,r-1\}$ *and* $B = \{0,1\}$. *Then, the size of the SMDD for an* N*-input* m*-output symmetric function* $R^N \rightarrow B^m$ *is at most*

$$\min_{k=1}^{N} \{m \cdot \sum_{i=0}^{k} \binom{i+r-1}{i} + \sum_{i=1}^{N-k} [2^{\binom{i+r-1}{i}} - 2]\}.$$

From here, we assume that $r = 4$.

Definition 2.2: *Let* ***wgt*** n *be an* n*-input* $(\lfloor \log_2 n \rfloor + 1)$*-output function that counts the number of 1's in the inputs, and represents it by a binary number, where* n *is the number of binary input variables, and* $\lfloor a \rfloor$ *denotes the largest integer not greater than* a. *It represents a* ***bit-counting function.***

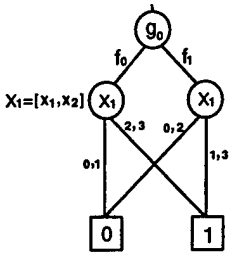


Figure 2.1: SMDD for functions $(f_0, f_1) = (x_1, x_2)$.

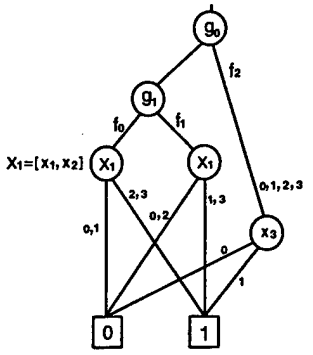


Figure 2.2: SMDD for functions $(f_0, f_1, f_2) = (x_1, x_2, x_3)$.

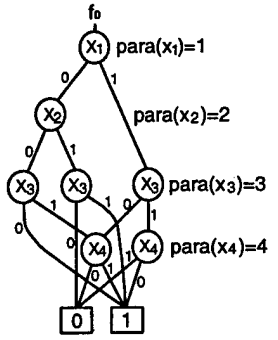


Figure 3.1: BDD for the function f_0.

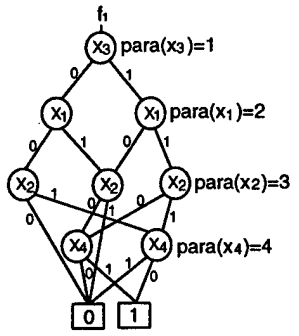


Figure 3.2: BDD for the function f_1.

Definition 2.3: *Let* ***inc*** *n be an n-input* $(n+1)$*-output function that computes* $x+1$*, where n is the number of binary input variables. It represents an* ***incrementing function.***

From the experimental results, we have the following conjectures for the bounds on the sizes of SMDDs for wgt n and inc n:

Conjecture 2.1: $size(SMDD, wgt\ n) = n\lfloor \log_2 n \rfloor + n - 2^{\lfloor \log_2 n \rfloor}$, *where* $n > 1$.

Conjecture 2.2: $size(SMDD, inc\ n) = 2n - 1$, *where* $n > 1$.

Theorem 2.3: *Let an SMDD represent m functions* $f_i = x_i$ $(i = 0, 1, \ldots, m-1)$*, where* x_i *is a binary input variable. Then, the size of the SMDD is m.*

Example 2.2: *Figs. 2.1 and 2.2 show the SMDDs for functions* $(f_0, f_1) = (x_1, x_2)$ *and* $(f_0, f_1, f_2) = (x_1, x_2, x_3)$*, respectively. Their sizes are 2 and 3, respectively.* ■

Note that the size of the SBDD to represent m functions $f_i = x_i$ $(i = 0, 1, \ldots, m-1)$ is also m [10].

3 Construction of Compact SMDDs

Compact SMDDs are important to represent multiple-output functions efficiently. In this section, we will show heuristic algorithms to optimize SMDDs. We consider the following approaches to reduce the sizes of SMDDs: 1) pairing of input variables; 2) ordering of input variables by sifting the 4-valued variables [16]; and 3) ordering of input variables by sifting pairs of 4-valued variables.

3.1 Pairing of Input Variables

Pairing of input variables is important to reduce the size of SMDDs. The input variables that greatly affect the size of the decision diagram are called ***influential.*** We will use a heuristic algorithm to pair the influential input variables of BDDs for multiple-output functions.

3.1.1 Our Method

In this part, we will show a heuristic algorithm to find good pairs of the input variables.

Definition 3.1: *Let* $U = \{u_1, u_2, \ldots, u_k\}$ *be a set of k variables. Let* $U_1 \subseteq U$*, and* $U_2 \subseteq U$*.* $P = \{U_1, U_2\}$ *is a* ***partition*** *of U iff* $U_1 \bigcup U_2 = U$*, and* $U_1 \bigcap U_2 = \phi$*.*

Example 3.1: *Let* $U = \{u_1, u_2, u_3, u_4\}$ *be a set of four variables. Then,* $\{[u_1, u_2], [u_3, u_4]\}$ *is a partition of U.* ■

Definition 3.2: *Let a BDD represent a function f.* $para(x_i)$ *is the* ***parameter of an input variable*** x_i ***at height i in the BDD with the variable ordering.*** *It denotes the level of the BDD for* x_i*. We assume that the smaller the value of* $para(x_i)$*, the more influential the variable* x_i*.*

Example 3.2: *Figs. 3.1 and 3.2 show the BDDs for the functions* f_0 *and* f_1*, respectively. The values of* $para(x_i)$ *for both BDDs are shown in the figures. For example,* x_1 *is the most influential variable in the BDD in Fig. 3.1, since* $para(x_1)$ *is the smallest among all* $para(x_i)$*.* ■

Definition 3.3: *Let* $F = (f_0, f_1, \ldots, f_{m-1})$ *be an n-input m-output function, and* $para_k(x_i)$ *be the parameter of the variable* x_i *in the BDD for* f_k*. Then,* $T = (T_1, T_2, \ldots, T_n)^t$ *is a* ***total parameter vector,*** *where* T_i *is calculated as follows:*

$$T_i = para(x_i) = \prod_{k=0}^{m-1} para_k(x_i).$$

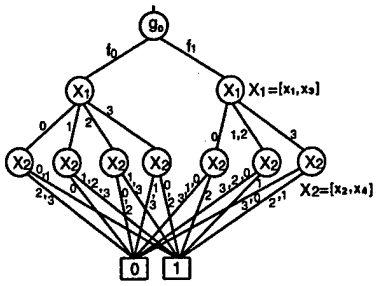


Figure 3.3: SMDD for the partition $\{[x_1, x_3], [x_2, x_4]\}$.

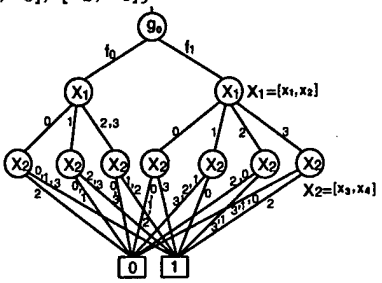


Figure 3.4: SMDD for the partition $\{[x_1, x_2], [x_3, x_4]\}$.

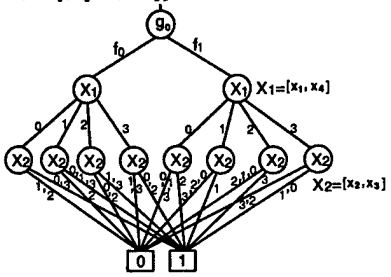


Figure 3.5: SMDD for the partition $\{[x_1, x_4], [x_2, x_3]\}$.

Example 3.3: *Consider the functions in Example 3.2. The total parameter vector for $F = (f_0, f_1)$ is*

$$T = \begin{matrix} \\ para(x_1) \\ para(x_2) \\ para(x_3) \\ para(x_4) \end{matrix} \begin{matrix} F \\ \begin{pmatrix} 2 \\ 6 \\ 3 \\ 16 \end{pmatrix} \end{matrix}.$$

■

Definition 3.4: *The* ***weight*** *$w(i,j)$ for a pair of input variables x_i and x_j $(i \neq j)$ is defined by*

$$w(i,j) = para(x_i) \cdot para(x_j).$$

Example 3.4: *In the functions of Example 3.2, the weights are as follows: $w(1,2) = 12$, $w(1,3) = 6$, $w(1,4) = 32$, $w(2,3) = 18$, $w(2,4) = 96$, and $w(3,4) = 48$.* ■

Algorithm 3.1: *(Pairing the input variables)*
Let $F : \{0,1\}^n \rightarrow \{0,1\}^m$. Let Q be a set of pairs of n input variables, and $q \in Q$. Let W be the list of sorted weights for the pairs of Q.

1. *Optimize the BDD for each output function f.*
2. *Calculate the total parameter vector T.*
3. *Calculate the weight $w(i,j)$ for each pair of input variables.*
4. *Make the list of weights, W sorted in ascending order.*
5. *Select $q \in Q$ with the smallest weight $w(i,j)$ from W, and eliminate the pairs that contain the input variables in q from Q. Update Q and W.*
6. *Repeat Step 5 until $Q \neq \phi$, and make a good partition of input variables with the selected pairs.*

Example 3.5: *Consider the functions in Example 3.2. There are three different ways of pairing four inputs:*
1)$\{[x_1, x_2], [x_3, x_4]\}$ *(SMDD in Fig. 3.4),*
2)$\{[x_1, x_4], [x_2, x_3]\}$ *(SMDD in Fig. 3.5), and*
3)$\{[x_1, x_3], [x_2, x_4]\}$ *(SMDD in Fig. 3.3).*
3) is a good partition of the input variables according to Algorithm 3.1, since $w(1,3)$ is the smallest element. ■

3.2 Ordering of Input Variables

The ordering of input variables is very important to reduce the sizes of SMDDs [2, 16, 17, 18, 19]. Sifting is an efficient method to find a good ordering of the input variables. Normal sifting [16] moves a single variable at a time, while group siftings [17, 18] move more than one variable at a time. Group siftings are faster than normal siftings to produce compact DDs. Pair sifting is one kind of group siftings that moves a pair of symmetric variables at a time. In this part, we will consider the normal sifting and the pair sifting of 4-valued input variables. In the case of the pair sifting, we find good pairs of the 4-valued input variables from MDDs. Functions are usually multiple outputs, and it is not so easy to find good pairs of the input variables for all the outputs. We use Algorithm 3.1 to find good pairs of the input variables.

Algorithm 3.2: *(Construction of an SMDD using the normal sifting)*

1. *Construct the SMDD by Algorithm 3.1.*
2. *Use the sifting algorithm* [16] *for the 4-valued input variables such that the size of the SMDD is minimized.*

From now, we will call this optimized SMDD as the SMDD with the normal sifting.

Algorithm 3.3: *(Construction of an SMDD using the pair sifting of 4-valued variables)*
Let $F : \{0,1\}^n \rightarrow \{0,1\}^m$.

1. *Construct the MDDs for F by Algorithm 3.1.*
2. *Find good pairs of 4-valued variables from the MDDs using the similar techniques to Algorithm 3.1, and make an initial variable ordering with the pairs.*
3. *Select a pair of 4-valued variables from the initial ordering, and use the sifting algorithm to find the position of the pair such that the size of the initial SMDD is minimized.*
4. *Repeat Step 3 until all the pairs from the initial ordering have been checked, and choose the smallest SMDD.*

Example 3.6: *Let* $\{[X_1, X_3], [X_2, X_4]\}$ *be a good partition of 4-valued input variables obtained by Algorithm 3.1. Then, the initial variable ordering for the pairs is* (X_1, X_3, X_2, X_4). ■

From now, we will call this optimized SMDD as the SMDD with the pair sifting.

4 Experimental Results

We implemented C programs to construct SBDDs and SMDDs. The benchmark functions were generated in two ways: 1) functions with 2-valued inputs for SBDDs; and 2) functions with 4-valued inputs for MDDs and SMDDs generated from the 2-valued ones. The encodings to the 4-valued inputs were done by using Algorithm 3.1. In the case of odd number of inputs, one input variable was remained in the 2-valued form for MDDs and SMDDs. We used the sifting algorithm [16] for input variables to reduce the sizes of SBDDs and SMDDs. In addition, we considered an extension to the sifting that moved pairs of 4-valued input variables to produce compact SMDDs. The symbol "†" in Tables 4.1 and 4.2 denotes the function with ***don't cares***, where the ***don't cares*** were set to zero during the experiment. Table 4.1 compares the sizes of SBDDs and SMDDs for some benchmark functions. Note that the nodes for output selection variable are not included in the size. We used Algorithms 3.1, 3.2, and 3.3 to optimize SMDDs, and chose the smaller ones between the SMDDs with the normal sifting and the SMDDs with the pair sifting. The results are shown in the column headed with ***our method***. Table 4.1 showed that, for many benchmark functions,

$$size(SMDD,\ F) < \tfrac{1}{2}\ size(SBDD,\ F).$$

To show the effectiveness of Algorithm 3.1, we also compared our results with the sizes of SMDDs for the best, average, and worst cases. For $n = 2t$ 2-valued input variables, there exist $Q = \frac{(n!)}{(t!)2^t}$ different ways of pairing the inputs. For example, when $n = 12$, we have $Q = 10395$ different pairings. We derived all possible SMDDs for different pairings to find the smallest one. The values in the column headed with ***best*** were generated in this way. We used the sifting algorithm [16] to order the input variables. Also, the columns headed with ***average*** and ***worst*** were obtained at the same time. Table 4.1 shows that, in many cases, our algorithms obtained SMDDs that are close to the best ones.

Table 4.2 compares the sizes of SMDDs obtained by two different methods. The column headed with ***our method*** was obtained in the same way as the column ***our method*** in Table 4.1. This table shows that our method often produces more compact SMDDs than [2].

Table 4.3 compares the sizes of MDDs and SMDDs for larger benchmark functions. In the MDDs of a multiple-output function, each MDD was optimized independently. So, MDDs may have different partitionings of the 2-valued inputs, but have the same ordering of the 2-valued input variables. However, nodes in the different MDDs are not shared even if they represent the same function. The variable ordering for the MDDs was obtained from the corresponding optimized SMDD. The column headed with ***sift*** denotes the sizes of SMDDs which were obtained by using the sifting of 4-valued input variables (Algorithm 3.2). On the other hand, the column headed with ***psift*** denotes the sizes of SMDDs which were obtained by using the sifting of pairs of 4-valued input variables (Algorithm 3.3). The ***ratios*** in the last column denote the sizes of SMDDs with the pair sifting to the sizes of SMDDs with the normal sifting. The CPU time for the pair sifting in the column 8 includes not only the time for the variable ordering, but also the time for finding the pairs of 4-valued input variables from MDDs. CPU time for both siftings in Table 4.3 shows that, the pair sifting is faster than the normal sifting. Experimental results show that MDDs are usually larger than SMDDs. Sometimes, MDDs are much larger than SMDDs. For example, the size of the MDD for e64 is 997, while the size of the SMDD for e64 is 110. In addition, Table 4.3 shows that, for many benchmark functions, SMDDs with the pair sifting are smaller than SMDDs with the normal sifting. For accpla, apex1, apex5, mish, seq, x1, and xparc, SMDDs with the normal sifting are smaller than SMDDs with the pair sifting. Sometimes, the both siftings have the same size, e.g. e64. Note that apex5 is one of the most complex benchmark functions in Table 4.3, and our pro-

Table 4.1: Comparison of the sizes of SBDDs and SMDDs.

Function name	In	Out	SBDD	SMDD			
				Our method	best	average	worst
alu2†	10	8	70	40	32	49.9	63
br1	12	8	85	37*	37	60.8	71
dc2	8	7	72	29*	29	38.5	50
dist	8	5	158	85	56	102.6	130
f51m	8	8	76	33*	33	46.7	62
fout	6	10	141	79	47	96.3	112
max512	9	6	184	65*	65	121.8	153
misex1	8	7	44	20	15	24.9	32
mlp4	8	8	150	70	53	114.2	138
newapla2	6	7	24	8*	8	16.7	21
t3	12	8	68	26*	26	39.4	55
z5xp1	7	10	79	38	34	48.6	60

In: number of inputs; Out: number of outputs.
*Our method obtained the best solution.
†function with ***don't cares***.

Table 4.2: Comparison of the sizes of SMDDs obtained by two different methods.

Function name	In	Out	SBDD	SMDD	
				Our method	Sasao [2]
alu2†	10	8	70	40	44
alu4	14	8	525	365	389
apla†	10	12	115	73	67
bw†	5	28	137	60	69
dk27†	9	9	35	24	22
duke2	22	29	366	297	281
misex1	8	7	44	20	26
misex2	25	18	100	69	80
misex3	14	14	557	310	322
rd53	5	3	23	11	11
rd73	7	3	45	17	17
rd84	8	4	60	24	24
sao2	10	4	85	43	49
sqr8	8	16	250	118	132
tial	14	8	701	355	380
vg2	25	8	90	62	50

Table 4.3: Comparison of the sizes of MDDs and SMDDs (SMDDs with the normal sifting and SMDDs with the pair sifting).

Function name	In	Out	MDD	SMDD				
				sift	time[1]	psift	time[2]	ratio[3]
accpla	50	69	1623	1065	74.25	1134	52.10	1.06
apex1	45	45	1845	1109	56.90	1245	35.15	1.12
apex5	117	88	1084	527	160.71	582	115.38	1.10
e64	65	65	997	110	38.10	110	25.32	1.00
ex4	128	28	1936	1487	35.44	1435	20.57	0.96
misg	56	23	107	55	1.66	47	0.73	0.85
mish	94	43	165	103	4.95	127	1.16	1.23
opa	17	69	519	235	0.15	178	0.03	0.75
seq	41	35	1262	730	12.25	812	9.15	1.11
soar	83	94	1053	612	6.26	520	4.71	0.84
x1	51	35	981	776	29.46	790	21.85	1.01
x2dn	82	56	315	142	5.13	101	4.05	0.71
x3	135	99	1185	715	112.50	682	87.35	0.95
x7dn	66	15	1016	490	15.02	351	10.61	0.71
xparc	41	73	2644	1012	65.70	1201	43.50	1.18

1. CPU time in sec on a JU1/170 to make variable orderings for SMDDs using the normal sifting.
2. CPU time in sec on a JU1/170 to make variable orderings for SMDDs using the pair sifting.
3. ratio = $\frac{size(SMDD)\ using\ the\ pair\ sifting}{size(SMDD)\ using\ the\ normal\ sifting}$.

gram required 8.52 CPU seconds to convert the 2-valued inputs into the 4-valued ones by using Algorithm 3.1 on a JU1/170 with 160 MB of main memory (Sun Ultra1-170 compatible).

From the experimental results (tables are not included), we also obtained the sizes of SBDDs and SMDDs to represent wgt n (Definition 2.2) and inc n (Definition 2.3). We observed that, for $2 \leq n \leq 11$,

$$size(SBDD, wgt\ n) = n^2 - n + \lfloor \log_2 n \rfloor + 1,$$
$$size(SMDD, wgt\ n) = n\lfloor \log_2 n \rfloor + n - 2^{\lfloor \log_2 n \rfloor},$$
$$size(SBDD, inc\ n) = 3n - 2, \text{ and}$$
$$size(SMDD, inc\ n) = 2n - 1,$$

where n is the number of 2-valued input variables, and $\lfloor a \rfloor$ denotes the largest integer not greater than a. In addition, we observed that, for $n > 4$,

$$size(SMDD, wgt\ n) < \frac{1}{2} size(SBDD, wgt\ n),$$

and for $n > 10$

$$size(SMDD, wgt\ n) < \frac{1}{3} size(SBDD, wgt\ n).$$

5 Conclusions and Comments

In this paper, we proposed a method to represent multiple-output functions using shared multiple-valued

decision diagrams (SMDDs). We also presented algorithms to pair the input variables of binary decision diagrams (BDDs), and to find good orderings of the multiple-valued variables in the SMDDs. We derived the sizes of SMDDs for general functions and symmetric functions. Experimental results showed that, in many cases, the size of the SMDD is a half that of the shared binary decision diagram (SBDD). For many benchmark functions, SMDDs with the pair sifting are smaller than SMDDs with the normal sifting. We also compared our method with other one. Algorithms 3.1 and 3.3 can be extended to group k input variables, where $k > 2$. SMDDs are useful in many applications such as design of multiplexer-based networks and design of pass-transistor logic networks [9, 13, 14].

Acknowledgements

This work was supported in part by a Grant in Aid for Scientific Research of the Ministry of Education, Science, Culture, and Sports of Japan.

References

[1] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Transactions on Computers*, 1986, C-35, (8), pp. 677-691.

[2] T. Sasao and J. T. Butler, "A method to represent multiple-output switching functions by using multi-valued decision diagrams," *Proc. IEEE International Symposium on Multiple-Valued Logic,* 1996, pp. 248-254.

[3] T. Sasao and J. T. Butler, "A design method for look-up table type FPGA by pseudo-Kronecker expansion," *Proc. IEEE International Symposium on Multiple-Valued Logic*, 1994, pp. 97-106.

[4] S. Minato, N. Ishiura, and S. Yajima, "Shared binary decision diagram with attributed edges for efficient Boolean function manipulation," *Proc. 27th ACM/IEEE DAC*, 1990, pp. 52-57.

[5] D. M. Miller, "Multiple-valued logic design tools," *Proc. IEEE International Symposium on Multiple-Valued Logic,* 1993, pp. 2-11.

[6] D. M. Miller and R. Drechsler, "Implementing a multiple-valued decision diagram package," *Proc. IEEE International Symposium on Multiple-Valued Logic,* 1998, pp. 2-11.

[7] Hafiz Md. Hasan Babu and T. Sasao, "Shared multi-terminal binary decision diagrams for multiple-output functions," *IEICE Trans. Fundamentals,* 1998, vol. E81-A, no. 12, pp. 2545-2553.

[8] Hafiz Md. Hasan Babu and T. Sasao, "Design of multiple-output networks using time domain multiplexing and shared multi-terminal multiple-valued decision diagrams," *Proc. IEEE International Symposium on Multiple-Valued Logic*, May, 1998, pp. 45-51.

[9] Hafiz Md. Hasan Babu and T. Sasao, "Time-division multiplexing realizations of multiple-output functions based on shared multi-terminal multiple-valued decision diagrams," *IEICE Trans. Inf. & Syst.,* May 1999 (to be published).

[10] Hafiz Md. Hasan Babu and T. Sasao, "Representations of multiple-output functions by binary decision diagrams for characteristic functions," *Proc. The Eighth Workshop on Synthesis And System Integration of MIxed Technologies (SASIMI'98)*, 1998, pp. 101-108.

[11] G. Epstein, *Multiple-Valued Logic Design: An Introduction*, IOP Publishing Ltd., London, 1993.

[12] M. Kameyama and T. Higuchi, "Synthesis of multiple-valued logic networks based on tree-type universal logic module," *IEEE Transactions on Computers*, 1977, C-26, (12), pp. 1297-1302.

[13] A. Thayse, M. Davio, and J-P. Deschamps, "Optimization of multi-valued decision algorithms," *Proc. IEEE International Symposium on Multiple-Valued Logic,* 1978, pp. 171-178.

[14] K. Yano, Y. Sasaki, K. Rikino, and K. Seki, "Top-down pass transistor logic design," *IEEE Journal of Solid-State Circuits*, 1996, vol. 31, no. 6, pp. 792-803.

[15] P. C. McGeer, K. L. McMillan, A. Saldanha, A. L. Sangiovanni-Vincentelli, and P. Scaglia, "Fast discrete function evaluation using decision diagrams," *Proc. International Workshop on Logic Synthesis*, 1995, pp. 6.1-6.9. Also, in *Proc. International Conference on Computer-Aided Design,* 1995, pp. 402-407.

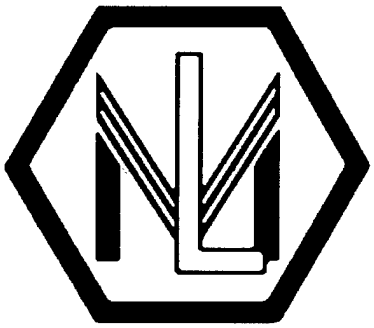
[16] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," *Proc. International Conference on Computer-Aided Design,* 1993, pp. 42-47.

[17] S. Panda and F. Somenzi, "Who are the variables in your neighbourhood," *Proc. International Conference on Computer-Aided Design*, 1995, pp. 74-77.

[18] C. Meinel and A. Slobodová, "Speeding up variable reordering of OBDDs," *Proc. International Conference on Computer Design*, 1997, pp. 338-343.

[19] H. Fujii, G. Ootomo, and C. Hori, "Interleaving based variable ordering methods for ordered binary decision diagrams," *Proc. International Conference on Computer-Aided Design*, 1993, pp. 38-41.

SESSION VIB
CIRCUITS II
CHAIR: Michitaka Kameyama



Ternary Multiplication Circuits Using 4-Input Adder Cells and Carry Look-Ahead

Andreas Herrfeld and Siegbert Hentschke
Institut für Periphere Mikroelektronik
University of Kassel, Wilhelmshöher Allee 71, 34109 Kassel, Germany
ah@digi.e-technik.uni-kassel.de / http://www.uni-kassel.de/fb16/ipm/dt/index.html

Abstract

We introduce a new implementation of a ternary adder with four inputs and two outputs. This ternary adder reduces the number of digits in a multiplication compared with a binary multiplication. One advantage of the ternary adder is that four instead of three inputs within a binary representation will be summed up. In this paper we will compare the complexity of binary against ternary multipliers. Timing diagrams will be given for the binary and the ternary case with an optimal order of the adder inputs. Finally, we present a ternary carry look-ahead circuit for a further reduction of total time delay.

1. Introduction

In this paper we address the problem of multiple-valued multiplications. First, we will give an overview about recent approaches to optimize the multiplication procedure. In the 1950's first investigations for the use of quasi-ternary number representations were made. The basic idea was presented by Booth in 1951 [1]. His idea was to combine two binary digits and to recode these into *signed* digits. The aim of his work was to develop a uniform algorithm for multiplication of two signed binary numbers. The disadvantage consists in an unfavorable behaviour, if sequences of "01" occur. In this case the recoded quasiternary number has more covered digits than the original one. A decisive alteration in the recoding structure from two to three bits by MacSorley [11] led to the fact, that the modified-Booth recoding forms the most common recoding structure for binary multipliers. It is an advantage of this recoding algorithm that at most half of the digits can be covered. By a corresponding design, taking these properties into account, this leads to a considerable saving in partial products when using it in a multiplier. In spite of the quasi*ternary* recoding, the realization in a hardware is carried out in established binary logic [5].

In parallel, circuits using multiple-valued logic (MVL) were developed [7]. In recent years, a number of papers combining MVL-techniques and recoding algorithms for the development of multipliers have been presented. Such a combination was given by Chen and Rajashekhara [2] in CMOS technique. But, the complexity for realizing the multiplier is very high, due to the fact that the chosen radix is 2 and only adders with two inputs are used. A mixed multiple-valued / binary approach was presented by Etiemble and Navi using current mode circuits [4]. By special 3BC-cells (3-valued to binary current-mode converters), ternary inputs will be recoded to binary outputs that will be summed up to multiple-valued currents. While all approaches presented so far work with radix 2 instead of higher radixes, Kameyawa et al. [10] gave an algorithm for multiplication in radix 4. Also the circuits of Ishizuka et al. [9] operate to radix 4 using current mode circuits and a radix-4 redundant number system. Ibrahim and Abdul-Karim [8] developed a radix 3 2x2 trit multiplier in CMOS. Besides the fact that the introduced adders are too extensive, they use the ternary digital COSMOS technique, introduced by Mouftah and Jordan [12]. But, the usage of resistors is of no significance for todays VLSI-realizations.

None of the approaches presented makes use of all advantages of the MVL-technique. On the one hand, this is based on the usage of a redundancy in the number representation in combination with a suboptimal radix. On the other hand, instead of an optimal radix, the possible number of inputs for an adder will not be exhausted. An early attempt with an optimal combination of number of adder-inputs and radix was done by Vranesic and Hamacher in ternary multi-threshold logic [13]. The ternary multiplier worked directly to radix 3. The introduced adder consists of four inputs and accomplished a four-to-two reduction of summands. Since every output has to be connected to an input of a following adder - providing that the output is not part of the result - this extension leads to a considerable saving of adders compared to a binary technique. A promising approach was presented by De and Sinha [3] in I^2L-technique. Besides an optimal combination of number of inputs and radix, they move on by parallelism through precarry operations that speed up the multiplication. In this paper we present ternary multipliers using optimal radix 3, a non-redundant encoding scheme and adders with up to four inputs.

0-7695-0161-3/99 $10.00 © 1999 IEEE

In the following section we will derive some equations needed for the estimations in the later sections. In section 3 the sum- and the carry-circuit of a ternary 4-input adder as well as a simulation result is presented. Section 4 compares the complexity of multiplication circuits using different kinds of adders. The timing behaviour of the resulting multiplication circuits is analyzed in section 5. For further improvements, ternary carry look-ahead circuits are presented in section 6. The main results are finally summarized in the conclusion.

2. Basic relations

In the following we assume that both, multiplicand and multiplicator U and V, have the same wordlength n. This is no common limitation, but simplifies the derived equations. Both numbers, U and V, have the representation

$$\begin{aligned} U &= u_{n-1}u_{n-2}\ldots u_1u_0 \\ V &= v_{n-1}v_{n-2}\ldots v_1v_0 \end{aligned} \tag{1}$$

whereby u_i and v_i can take the values 0 and ± 1. The representation of U and V in (1) is assumed to radix 3. Each bit position i of U and V corresponds to a so-called *weight* which is 3^i in the ternary case. For the corresponding decimal value follows

$$\begin{aligned} U &= \sum_{i=0}^{n-1} u_i 3^i \\ V &= \sum_{i=0}^{n-1} v_i 3^i \end{aligned} \tag{2}$$

The representation in (2) illustrates that the ternary expression is easier than a binary one, due to the ternary values 0 and ± 1. The multiplication of U and V ($M=U\cdot V$) leads to equation (3) [3]

$$M = \sum_{i=0}^{n-1} 3^i \sum_{j=0}^{i} u_{i-j}v_j + \sum_{i=n}^{2n-2} 3^i \sum_{j=i-n+1}^{n-1} u_{i-j}v_j \tag{3}$$

The multiplication of u_iv_j will be called a *partial product pp(i+j)*. For the ternary case this partial product can only lie within the range -1 to +1. Usual adders sum up several inputs of the same weight and produce two outputs, one at the same and one at the next higher bit position. As we work to radix 3, the result can lie within the range -4 to +4. This means, however, that a ternary adder can process four inputs of the same bit position in contrast to three inputs of a binary adder. A further advantage of the ternary number representation is the treatment of the sign, because it causes no additional expense. Desisting from the recoding algorithms, mentioned in the introduction, in binary technique we have to spend one bit more to accommodate the sign. By consideration of the sign bit, we can derive the relation for a binary representation with the same resolution as a ternary representation to be

$$b = n\cdot\frac{\log(3)}{\log(2)} + 1 \approx n\cdot 1.58 + 1 \tag{4}$$

whereby b gives the number of digits of a binary representation. With increasing n, the number of partial products in a multiplication increases naturally. Independent of the radix, *pp(i)* partial products

$$pp(i) = \begin{cases} i+1 & \text{for } 0\le i\le n-1 \\ 2n-i-1 & \text{for } n\le i\le 2n-2 \end{cases} \tag{5}$$

have to be processed for every bit position. Summed up from the starting index 0 to the complete wordlength of the $n\cdot n$ multiplication result $2n-2$ it follows

$$PP = \sum_{i=0}^{2n-2} pp(i) = n^2 \tag{6}$$

The higher number of digits of a binary representation consequently influences the number of partial products PP quadratically. On the other hand, the expense of ternary gates is much higher than that of binary ones. Firstly, the following analysis will show in detail, whether the ternary technique is advantageous in contrast to the binary one or not. For this analysis, we have to develop a ternary adder with four inputs and two outputs. Secondly, we will analyze the effects of a reduction from k inputs to l outputs with $k\ge l$ for adders with k=2 to 4 inputs. Since every output of an adder has to be connected with an input of a following adder - except this output is part of the result - it follows a nonlinear relation between the number of adders and the wordlength n. This relation should be analyzed in dependence on the types of adders. With this analysis we can compare the complexity of both binary and ternary multipliers.

3. TDDNL 4-input adder

As a ternary switching technique we have choosen the ternary dynamic differential no race logic (TDDNL) technique [6]. The realization of logical operations with one or two inputs as well as connection circuits to the binary system are given in [6]. The design of a TDDNL-adder with three inputs is also shown there. This design should be expanded to four inputs.

TDDNL-gates use a logical array of nmos-transistors with three outputs, f_n, f_z, and f_p, i.e., a negative, zero, and a positive output. In the evaluation phase, exactly one of these three outputs must be connected with the source M. Thus, the development of optimal arrays is an optimization problem that can be solved by existing software tools. In contrast to that, the presented method for designing the logic arrays is heuristically. With this heuristical method we get optimal solutions for the derived adder as well as for the carry look-ahead gate presented in section 6.

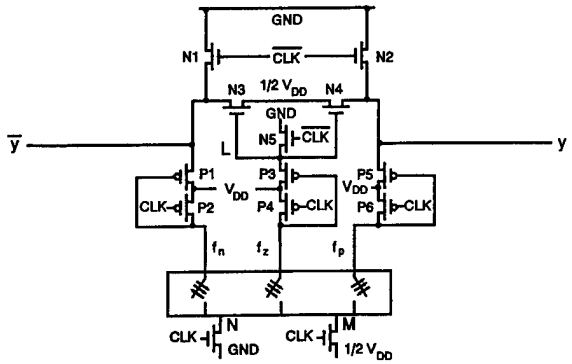


Fig. 1: Driver using ternary dynamic differential no race logic (TDDNL) from [6]

The heuristical method for the determination of the logical array, with every variable used having a fixed weight, can be described as follows:

Each variable will be worked out in a step. Firstly, if variables have differing weights, we have to decide, which variable is to be used first. If all variables are equal in weight, the order of the variables is of no importance. In the first step, all combinations of the first variable will be covered. All nodes arising from this step will be combined with all three combinations of the next variable. In the following, the actual sums will be written to the new arising nodes. The sums will be calculated by the sum of the two variables processed up to this point. All nodes with the same sum can be connected together. For a further simplification attention must be paid to the fact, whether a node eventually represents a fixed result. That means that the result is not depending on the following variables. In this case, this node can be connected to the corresponding output. All other nodes will be processed in the same manner as described, until all variables have been used.

The following development of a 4-input sum- and carry-circuit elucidates this method. In case of a ternary adder with four inputs the four variables *w, x, y,* and *z* have to be processed. The order of the variables is of no significance because the four inputs have the same weight. After each node, we will form a sum that is composed of the complete path to the source-node *M*. In the case of a ternary carry-signal in fig.2, the sum after the first variable *w* at the left node is -1, at the middle node it is 0 and at the right node it is +1. These three nodes will be combined with the three combinations of *x*. The sums at the left node connected with $\bar{x}$ will be -2, with x^0 -1 and connected with x it will be 0. After this procedure is done for all 9 cases, we look for nodes with equal weights. In the case of the carry signal we can reduce the nine nodes to five, according to four cases where the sums are equal. The connection of the nodes means that the result is independent on the way, *how* this sum is obtained. Especially, an intermediate sum of 1 is independent how it comes about, either by the combination w=1 / x=0 or by w=0 / x=1. The number of nodes that require further processing is minimal with this method. Every one of the five nodes after *x* and *w* with the sums in the range from -2 to 2, must be combined with the three combinations of *y*. The connection of nodes with the same weight after *y*, results in seven nodes in the range from -3 to +3. Because of a negative carry having to be set if the sum is equal or less than -2, the node with the sum -3 can be connected with the output Ca_n in fig.2. Similar considerations eliminate the node with the sum +3. The remaining five nodes have to be combined with all combinations of *z*. The logical array obtained by this method has a least number of transistors for this problem. As a further property of this method, no states will be propagated back in the network that could cause a wrong result (compare fig.12 in [6]).

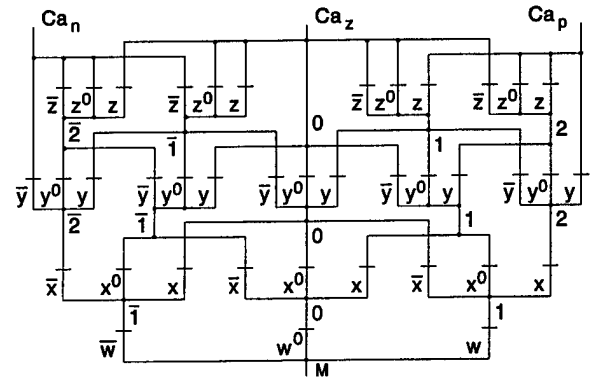


Fig. 2: Logical array of a ternary carry-signal

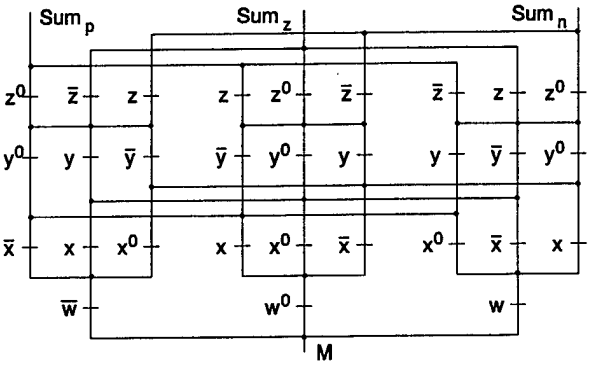


Fig. 3: Logical array of a ternary sum-signal

The logical array for the sum is still more favorable than that of the carry, because the modulo-3-operation takes care for the property that the intermediate sums

cannot exceed the range -1 to +1. Regardless of the number of variables used, almost three nodes after every step have to be processed further. Both logical arrays, for the sum and for the carry, are given in fig.2 and 3. The carry-signal needs 39 nmos-transistors, whereby the sum-circuit uses only 30 transistors. These circuits have to be expanded by the TDDNL-driver in fig.1.

4. Complexity of multipliers

We can not explain our analysis in detail at this point. But, the results in fig.5 show significant advantages of ternary multiplications against binary ones.

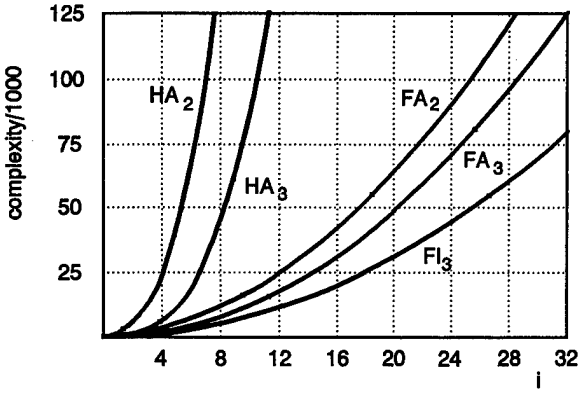


Fig. 4: Complexity of multipliers with n digits to base 3; half adder HA: 2 inputs; full adder FA: 3 inputs; four input adder FI: 4 inputs; index is radix

For the detailed analysis, we have made the same assumptions as in [6]. According to this analysis the following complexity results for the different adder types: binary adder with 2 inputs: complexity 54 and 30 transistors; binary adder with 3 inputs: complexity 60 and 36 transistors; ternary adder with 2-inputs: complexity 90 and 58 transistors; ternary adder with 3 inputs: complexity 127 and 91 transistors; ternary adder with 4 inputs: complexity 159 and 119 transistors.

Fig.4 shows the relations, standardized to the number of digits n as an exponent to base 3. For n=32, e.g., a resolution of more than $1.853 \cdot 10^{15}$ follows for the ternary case, whereby in a binary representation according to (4) nearly 52 bit have to be provided. From the above formulas a complexity of 79200 results for a 32-trit-multiplier. This is equal to 60032 transistors.

5. Timing behaviour

Fig.4 reflects a comparison with respect to the complexity of binary and ternary multipliers. Regarding to the timing behaviour we can expect the ternary multiplier to be superior in contrast to binary multipliers, because considerably less steps have to be processed up to the final result. With respect to the switching delays, TDDNL-gates can be compared to have a similar delay in contrast to binary dynamic gates. This is obvious, if we consider the logic block in fig.2 that is built by n-channel MOSFETs only, whereby a maximal number of four transistors is connected in series. Besides a faster switching property against p-channel MOSFETs, this means a lower input capacity too. For the TDDNL-gates only the propagation delay of a TDDNL-driver (fig.1) has to be added. But, this delay is formed by the switching of only one p-channel transistor.

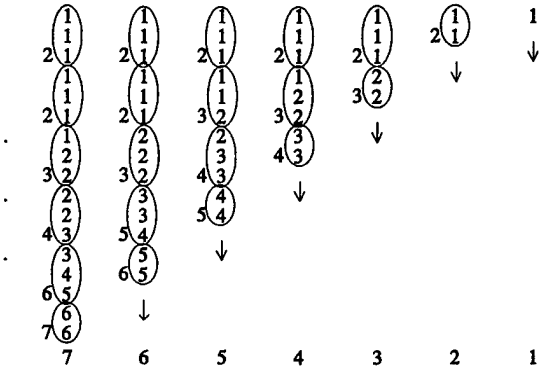


Fig. 5: Propagation delay according to an optimal ordering of summands in a binary multiplier (τ=1)

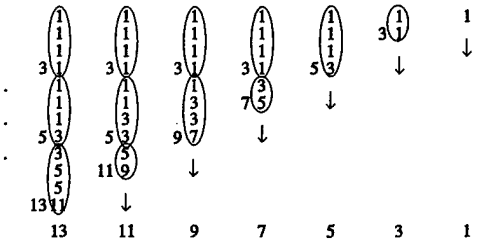


Fig. 6: Propagation delay according to an optimal ordering of summands in a ternary multiplier (τ=2)

For a more detailed analysis, we will make the following assumptions: All partial products will be formed in a standardized delay time τ=1. The propagation delay for a binary dynamic gate will also be τ=1 from every input to both the sum- and the carry-output. To demonstrate the advantages of the ternary technique, we assume the TDDNL-gates to have a propagation delay of τ=2 from any input to any output.

In order to optimize the circuit regarding to its timing behaviour, we will introduce a new order technique. This technique optimizes the timing behaviour, but complicates the layout of the complete circuit. However, we will keep

this in the background since efficient algorithms exist for solving these problems. The new order technique functions as follows:

All partial products will be ordered according to their magnitudes for each weight and will be written into a list. An adder processes the 2, 3 or 4 (in proportion to the adder used) fastest products and forms two output signals, called intermediate sums is(i). The magnitude of these outputs is equal to the largest magnitude of the inputs plus the standardized propagation delay as specified. The used inputs are cancelled from the list and the new two magnitudes are written into the list, which will be ordered again. This procedure will be processed from the LSB to the MSB, until only one entry remains for every weight.

Fig.5 shows the behaviour for adders with three inputs (a) which form an input-to-output reduction from 3 to 1/1 (sum/carry). Fig.6 shows the behaviour with four input adders which form an input-to-output reduction from 4 to 1/1. The numbers on the left hand of the ellipses show the calculated propagation delay that has to be written to the same and to the next higher weight. This number has to be calculated by adding the time delay τ to the biggest number in the ellipse.

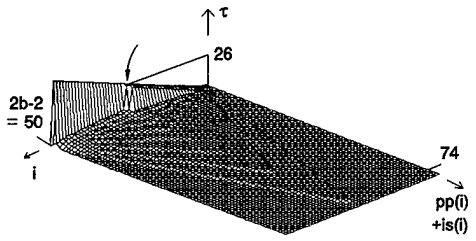


Fig. 7: Timing behaviour of a 26 bit binary multiplier

We observe that the propagation delay of a ternary multiplier-result increases by two whereby that of the binary result increases by one per weight. But, the crucial difference is that we can use a carry look-ahead method for much lower weights in the ternary case in contrast to the binary case. This is based on the fact that only *one* intermediate sum has this high propagation delay. All other intermediate sums *is(i)* are stable much earlier. Thus, a carry look-ahead technique can be used in the ternary case from the lowest bits up to the MSB of the result.

Fig.8 shows the behaviour of a ternary 16-trit multiplier in contrast to a binary 26-bit multiplier in fig.7, which means nearly the same resolution for both cases. The arrows mark the weight from that onwards the usage of a CLA-circuit is sensible. Desisting from the carry-signal that is rippled through the complete wordlength, the largest standardized intermediate sum is 11 in the ternary case in contrast to 26 in the binary case. This is in spite of the fact that the estimated propagation delay of the ternary adder is twice as large as that of the binary adder. Furthermore, it has to be considered that the wordlength of the ternary result is reduced by a factor of 1.58 compared to the binary result.

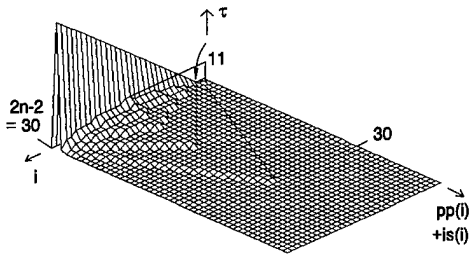


Fig. 8: Timing behaviour of a 16 TRIT ternary multiplier

The concluding chapter describes the development of the ternary CLA-circuits that can be used with the ternary multiplier circuits.

6. TDDNL carry look-ahead technique

TDDNL-gates have the property that the precharge-phase concludes with an invalid combination of the output. Whether the output has to be -1, 0 or 1, the TDDNL-gate has to change its state. Thus, the carry signal ripples through all stages connected in series, independent of the states of the variables. A following gate can switch only when the preceding gate has a valid output combination. As a result, the total propagation delay depends on the number of gates, connected in series. Thus, the aim of the CLA-logic is to speed up this procedure by connecting the function of two or more adders into one special structure.

The basic carry look-ahead principle was developed by Weinberger and Smith in 1956 [14] for binary logic. But, we refrain from the usage of generate- and propagate-signals as used in this early work, because this would require two additional TDDNL-gates. Instead, we will use the carry- and the sum-outputs of the circuits directly. For the development of a ternary carry look-ahead circuit we assume that 2 bits per weight have to be processed in a final adder.

Fig.9 shows the CLA-structure with 3 inputs $c_{in\,i}$, c_{in}, and s_i with the weight 1 and two inputs c_{i+1} and s_{i+1} with the weight 3. The logical switching circuit must connect the output $c_{out\,n}$ with the source in the case of a sum equal

or less than -5 (=-9+3+1). The output $c_{out\ p}$ has to be connected in the case of a sum equal or greater than +5 (=9-3-1). In the last case, the output $c_{out\ z}$ has to be connected with source, if the sum of the five inputs lies in the range -4 to +4. The development of the circuit is conformed to the heuristic method described earlier. To speed up the switching of this circuit, the time critical variable $c_{in\ i}$ is connected to the output as closely as possible.

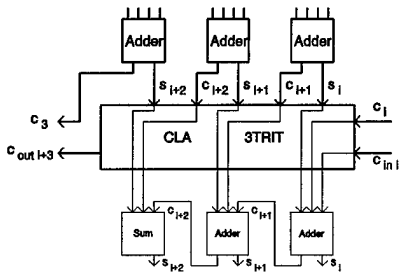


Fig. 9: Block diagram of a 3-TRIT CLA

Finally, fig.10 shows the complete logic of a 3-trit CLA-circuit. As an argument against such a structure one can remark that the switching delay of seven transistors in series is very high. But, six of the seven inputs are stable much earlier than the critical input $c_{in\ i}$.

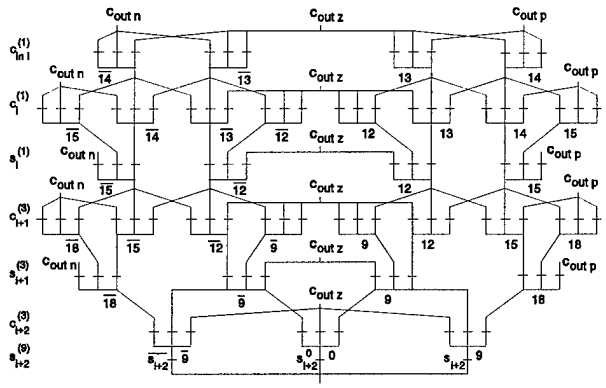


Fig. 10: Switching circuit of a 3-TRIT CLA, groups of 3 transistors connected with $\bar{x}$, x^0 und x

7. Conclusion

We have derived formulas for the number of adders in an $n \cdot n$ bit binary and ternary multiplication structure. The complexity for a binary array-multiplier is nearly twice as large as that for a ternary multiplier with a corresponding resolution. In addition, we have shown the advatages of the timing behaviour with an optimal order of partial products and intermediate sums. The presented ternary CLA-circuits can be used to speed up the processing of the final multiplication result.

References

[1] Booth, A.D., "A Signed Binary Multiplication Technique", *Quart. Journal of Applied Mathematics*, pp. 236-240, 1951.

[2] Chen, I.-S.E.; Rajashekhara, T.N., "A Fast Multiplier Design Using Signed-Digit Numbers and 3-Valued Logic", *Proceedings of the 33rd Midwest Symposium on Circuits and Systems*, pp. 881-884, 1991.

[3] De, M.; Sinha, B.P., "Fast Parallel Algorithm for Ternary Multiplication Using Multivalued I^2L Technology", *IEEE Transactions on Computers*, Vol.43, No.5, pp. 603-607, 1994.

[4] Etiemble, D.; Navi, K., "A Basis for The Comparison of Binary and m-Valued Current Mode Circuits: the Multioperand Addition with Redundant Number Systems", *Proceedings of the International Symposium on Multiple-Valued Logic*, pp. 216-221, 1993.

[5] Hentschke, S.; Herrfeld, A.; Reifschneider, N.; Förster, D.; Heinemann, M.; Wicke, R., "A Flexible Repetitive CSD Code Filter Processor Unit in CMOS", *Seventh Annual IEEE International ASIC Conference and Exhibit*, pp. 261-264, 1994.

[6] Herrfeld, A.; Hentschke, S., "Ternary Dynamic Differential No Race Logic", *International Journal of Electronics*, Vol.79, pp. 63-79, 1995.

[7] Hurst, S.L., "Multiple-Valued Logic - Its Status and Its Future", *IEEE Transactions on Computers*, Vol. C-33, No.12, pp. 1160-1179, 1984.

[8] Ibrahim, S.M.; Abdul-Karim., A.H., "2x2 Trit Ternary Logic Multiplier Design Employing CMOS Circuits", *Modelling, Simulation & Control*, Pt.A, pp. 27-36, 1990.

[9] Ishizuka, O.; Ohta, A.; Tanno, K.; Tang, Z.; Handoko, D., "VLSI design of a Quaternary Multiplier with Direct Generation of Partial Products", *Proceedings of the 27th International Symposium on Multiple-Valued Logic*, pp. 169-174, 1997.

[10] Kameyama, M.; Kawahito, S.; Higuchi, T., "A Multiplier Chip with Multiple-Valued Bidirectional Current-Mode Logic Circuits", *IEEE Computer*, pp. 43-56, 1988.

[11] MacSorley, O.L., "High-Speed Arithmetic in Binary Computers", *Proceedings of the IRE*, pp. 67-91, 1961.

[12] Mouftah, H.T.; Jordan, I.B., "Implementation of a 3-Valued Logic with COS-MOS Integrated Circuits", *Electronics Letters*, Vol.10, No.21, pp. 441-442, 1974.

[13] Vranesic, Z.G.; Hamacher, V.C., "Threshold Logic in Fast Ternary Multipliers", *Proceedings of the International Symposium on Multiple-Valued Logic*, pp. 373-387, 1975.

[14] Weinberger, A.; Smith, J.L., "A One-Microsecond Adder Using One Megacycle Circuitry", *IRE Transactions on Electronic Computers*, pp. 65-73, 1956.

Down Literal Circuit with Neuron-MOS Transistors and Its Applications

Jing Shen, Koichi Tanno, Okihiko Ishizuka
Miyazaki University, Miyazaki, 889-2192 Japan
shen@esl.miyazaki-u.ac.jp

Abstract

A voltage-mode neuron-MOS(νMOS) down literal circuit which realizes an arbitrary down literal function is proposed. It provides the benefit that the circuit can be easily fabricated by standard CMOS process, instead of the multi-level ion implantation applied in the conventional circuit. It has a variable threshold voltage by way of controlling only two bias voltages. Its noise margin and switching sensitivity are greater than those of variable-threshold C-νMOS inverter presented by Shibata. The threshold voltage errors of the circuit caused by device parameters mismatch is also analysed. Using the νMOS down literal circuits, literal and T-gate circuits are also presented. Performances of the proposed circuits are evaluated using HSPICE simulations with MOSIS 2.0μm CMOS device parameters.

1 Introduction

With the development of multi-valued logic (MVL) theory, the need for MVL circuit design is increased. There are two kinds of MVL circuit generally used: voltage-mode and current-mode. Current-mode MVL circuits are popular with many researchers for their advantage that the algebraic weighted sum or difference of input currents can be created easily, requiring inactive or passive components[1][2]. However, because they are current-mode, power consumption problem is serious. Conversely, voltage-mode MVL circuits have the advantage of low power consumption. But since they require the multi-level ion implantation process technology to realize multi-threshold voltages in MOS transistors, complex fabrication and high cost are inevitable [1]. In order to solve these problems, we try to use a general CMOS process to realize voltage-mode MVL circuits.

A neuron-MOS (νMOS) transistor is a kind of novel transistor device with multi-input gates. It was proposed by Shibata and Ohmi in 1992 [3]. The νMOS transistor is characterized by variable threshold voltage achieved by controlling the voltages of the multi-input gates. Applying this feature to voltage-mode MVL circuits, we present a νMOS down literal circuit in this paper. The proposed circuit has greater noise margin and switching sensitivity than Shibata's variable-threshold C-νMOS inverter that can also realizes down literal function in MVL [3].

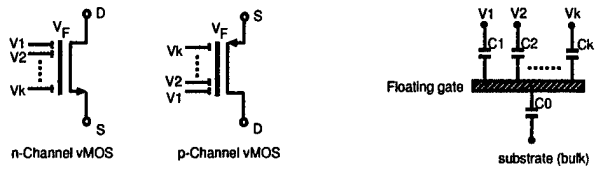
Basing on νMOS down literal circuits, literal and T-gate circuits can be easily composed. All the proposed circuits are verified by HSPICE simulations with MOSIS 2.0μm CMOS device parameters. Because νMOS can be fabricated by the standard CMOS process with a two-poly layer, the νMOS voltage-mode MVL circuits can be realized without the multi-level ion implantation process technology. The ease of fabrication will make voltage-mode MVL circuits more practically, and the development of MVL hardware will push the MVL research work forward.

2. νMOS down literal circuit

2.1. Neuron-MOS (νMOS) transistor

A νMOS transistor is the device composed of a floating-gate and multi-input gates that are capacitively interacting with the floating-gate. Figure 1 shows the illustration of νMOS transistor. C_0 is the oxide capacitance between the floating-gate and the substrate. In the case of a k-input n-channel νMOS, capacitances between the multiple input gates and the floating-gate are defined as $C_1, C_2 ... C_k$ in order from the drain side. When the floating-gate to source voltage (V_{FS}) is larger than the threshold voltage (V_T seen from the floating-gate, the νMOS switches ON, with the saturation drain-source current I_{DS} represented as Eq.(1). The corresponding transconductance parameter K, floating-gate voltage V_F and capacitive weight w_i are given by Eqs.(2-4) [2].

0-7695-0161-3/99 $10.00 © 1999 IEEE



(a) Symbolic representation (b) Capacitive model

Figure 1. Description of Neuron-MOS

$$I_{DS} = K(V_{FS} - V_T)^2 \quad , \tag{1}$$

$$K = \frac{1}{2}\mu C_{OX}\frac{W}{L} \quad , \tag{2}$$

$$V_F = \sum_{i=1}^{k} w_i V_i \quad , \tag{3}$$

$$w_i = \frac{C_i}{C_0 + \sum_{j=1}^{k} C_j} \tag{4}$$

where μ is the electron mobility, C_{OX} is the gate oxide capacitance per unit area, W is the channel width, L is the channel length, C_i is the capacitance between the floating-gate and ith input gate. The initial charge of the floating-gate is assumed to be zero.

The equation of I_{DS}(Eq.(1)) can be transformed to Eq.(5) which having the same form as that of a general MOS transistor.

$$I_{DS} = K^\star(V_j - V_S - V_{Tj}^\star)^2 \quad , \tag{5}$$

where,

$$K^\star = K w_j^2 \quad , \tag{6}$$

$$V_{Tj}^\star = \frac{1}{w_j}[V_T + (1 - w_j)V_S - \sum_{i \neq j}^{k} w_i V_i] \quad . \tag{7}$$

From Eq.(7), it is known that the equivalent threshold voltage $V_T^\star$ seen from j-th input gate can be varied by controlling the other input-gate voltages. Thereupon, in one chip MOS transistors with multiple threshold voltages can be realized by using νMOS, without multi-level ion implantation. Employing this feature in voltage-mode MVL circuit, we proposed the νMOS down literal circuit.

2.2. Circuit description

A down literal circuit is the fundamental element in voltage-mode MVL circuits. MVL logic functions are almost always developed from the down literal function, and most voltage-mode MVL circuits consist of

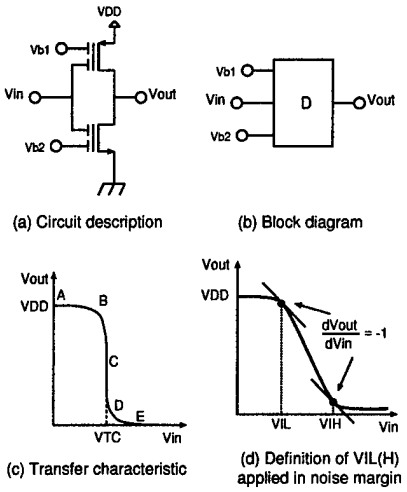


Figure 2. νMOS down literal circuit

down literal circuits. The down literal function is given by Eq.(8).

$$D_i(x) = \begin{cases} R-1 & x \leq i \\ 0 & x \geq i+1 \end{cases} \tag{8}$$

where $i \in \{0, 1, ...R-2\}$, $x \in \{0, 1, ...R-1\}$ in R-valued system [1]. The conventional voltage-mode CMOS down literal circuit is composed of a PMOS and an NMOS. Its structure is the same as the binary inverter. But the threshold voltages of PMOS and NMOS are to be fixed to different values according to "i" in Eq.(8). In order to fabricate down literal circuits in one chip, multi-level ion implantation is necessary to implement multi-threshold voltages of MOS transistors, which increases the complexity and cost of fabrication. To solve this problem, variable-threshold transistor-νMOS is considered to replace the general MOS transistor with fixed threshold voltage. A νMOS down literal circuit that can be fabricated by a standard CMOS process is proposed here.

The νMOS down literal circuit consists of one two-gate p-channel νMOS (νPMOS) and one two-gate n-channel νMOS (νNMOS) as shown in Fig.2(a). The common input to νPMOS and νNMOS, V_{in}, is taken as the input of the down literal circuit, and V_{b1}, V_{b2} are bias voltages(Fig.2(a)(b)). According to Eq.(7), the equivalent threshold voltage of νNMOS transistor ($V_{in}^\star$) seen from the input V_{in} is given by Eq.(9), and the equivalent threshold voltage of νPMOS ($V_{ip}^\star$) is given by Eq.(10) in a same way.

$$V_{tn}^\star = \frac{1}{w_{n1}}V_{tn} - \frac{w_{n2}}{w_{n1}}V_{b2} \quad , \tag{9}$$

$$V_{tp}^{\star} = \frac{1}{w_{p2}} V_{tp} - \frac{w_{p1}}{w_{p2}} V_{b1} - \frac{w_{p2}-1}{w_{p2}} V_{DD} \quad , \qquad (10)$$

where $w_{n(p)i}$, $i \in \{1,2\}$ are capacitive weights of νNMOS(νPMOS). Therefore, this νMOS down literal circuit can be taken as a common binary CMOS inverter with threshold voltage $V_{tn}^{\star}$ in NMOS and $V_{tp}^{\star}$ in PMOS . Because $V_{tn(p)}^{\star}$ can be varied by setting the threshold voltages V_{b1}, V_{b2}, the inverter has different threshold voltages, which satisfying the requirement of down literal circuit of MVL.

2.3. Transfer characteristic

The transfer characteristic of a down literal circuit is shown in Fig.2(c). $V_{DD} = (R-1)V_d$, V_d is unit voltage. When $V_{in} = xV_d$ is greater than the circuit threshold voltage $V_{TC} = (i + 0.5)V_d$, V_{out} turns from V_{DD} to 0. The key of MVL down literal circuit is that V_{TC} should have various values according to "i" in Eq.(8). In a manner analogous to a binary CMOS inverter, the transfer characteristic shown in Fig.2(c) is divided into 5 regions accroding to the operation of νMOS transistors. [4]

Region A: $0 \leq V_{in} \leq V_{tn}^{\star}$

In this region, νNMOS is OFF, there is no current flowing over the circuit, $V_{out} = V_{DD}$.

Region B: $V_{tn}^{\star} \leq V_{in} \leq V_{TC}$

In this region, νNMOS is in saturation region, and νPMOS in linear region.

Region C: $V_{in} = V_{TC}$

Both νNMOS and νPMOS operate in saturation region. Their saturation currents are I_p and I_n, respectively.

$$I_n = K_n^{\star}(V_{in} - V_{tn}^{\star})^2 \quad , \qquad (11)$$

$$I_p = K_p^{\star}(V_{in} - V_{DD} - V_{tp}^{\star})^2 \quad , \qquad (12)$$

where transconductance parameters: $K_p^{\star} = K_p w_{p1}^2 = \frac{\mu_p}{2} C_{OXp} \frac{W_p}{L_p} w_{p1}^2$, $K_n^{\star} = K_n w_{n1}^2 = \frac{\mu_n}{2} C_{OXn} \frac{W_n}{L_n} w_{n1}^2$, $\mu_{p(n)}$ is the electron mobility, $C_{OXp(n)}$ is the gate oxide capacitance per unit area, $W_{p(n)}/L_{p(n)}$ is the channel width/length of νP(N)MOS. $V_{tp(n)}^{\star}$ is the equivalent threshold voltage of νP(N)MOS seen from V_{in}(Eqs.(9)(10)).

In a manner analogous to binary CMOS inverter, the down literal circuit threshold voltage V_{TC} is defined as V_{in} in this C region [4]. and it can be derived from $I_p = -I_n$.

$$V_{TC} = \frac{V_{DD} + V_{tp}^{\star} + V_{tn}^{\star}\sqrt{K_R^{\star}}}{1 + \sqrt{K_R^{\star}}} \quad , \qquad (13)$$

$$K_R^{\star} = \frac{K_n^{\star}}{K_p^{\star}} = \frac{K_n}{K_p} \frac{w_{n1}^2}{w_{p2}^2} \quad . \qquad (14)$$

According to Eqs.(9)(10)(13) and (14), if there are $K_n = K_p$, $|V_{tp}| = V_{tn}$ and $w_{n1,2} = w_{p1,2} = w = \frac{1}{2}$ (capacitances between input gates and floating gate are identical in νN(P)MOS, V_{TC} shown in Eq.(13) can be simplified to

$$V_{TC} = V_{DD} - \frac{V_{b1} + V_{b2}}{2} \quad . \qquad (15)$$

Here, the restriction of V_{b1}, V_{b2} is given by Eq.(16) which is derived from saturation condition of νMOS transistors in the circuit.

$$-4V_{tn} \leq V_{b1} - V_{b2} \leq 2V_{DD} - 4|V_{tp}| \quad . \qquad (16)$$

According to Eq.(15), V_{TC} of this circuit can be varied by controlling the values of the bias voltages(V_{b1}, V_{b2}), and different down literal functions can be implemented.

In this region, the peak current (I_{Peak}) is reached, and its value is

$$I_{Peak} = \frac{K_n}{8}(V_{TC} + V_{b2} - 2V_{tn})^2 \quad . \qquad (17)$$

It is the saturation current flowing over the νNMOS and νPMOS in Fig.2(a).

Region D: $V_{TC} < V_{in} \leq V_{DD} + V_{tp}^{\star}$

νPMOS is in saturation region, and νNMOS in linear region.

Region E: $V_{in} > V_{DD} + V_{tp}^{\star}$

In this region, νPMOS is cut-off, $I_p = 0, V_{out} = 0$.

From the analysis above, it is known that the νMOS down literal circuit operates like the general CMOS inverter. The νMOS down literal circuit can also be considered as a CMOS inverter in which threshold voltages of NMOS and PMOS are replaced by the equivalent ones: $V_{tn}^{\star}$(Eq.(9)) and $V_{tp}^{\star}$(Eq.(10)), respectively. The circuit threshold voltage V_{TC} is corresponding to the gate threshold voltage in the general inverter [4]. According to Eq.(15), V_{TC} of the proposed circuit can be varied by controlling the values of the bias voltages(V_{b1}, V_{b2}), therefore different down literal functions can be realized using one proposed circuit.

The mismatch of device parameter will lead to the error on V_{TC}. The error of V_{TC} is defined in Eq.(18).

$$E = V_{TC'} - V_{TC} \quad , \qquad (18)$$

where V_{TC}' is threshold voltage of a circuit with device parameter mismatch. The real value of parameter $*$ is $*'$.

(1) In the case of $w' \neq \frac{1}{2}$, $\Delta w = w' - \frac{1}{2}$

$$E_w = \frac{-2\Delta w}{1 + 2\Delta w} V_{DD} \quad . \qquad (19)$$

(2) In the case of $V'_{tn} \neq |V'_{tp}|$, $\Delta V_t = V'_{tn} - |V'_{tp}|$

$$E_{V_t} = \Delta V_t \ . \tag{20}$$

(3) In the case of $K'_R \neq 1, \Delta K_R = K'_R - K_R$

$$E_{K_R} = \frac{(\sqrt{1+\Delta K_R}-1)^2}{2\Delta K_R}(4V_{tn} - 2V_{DD} + V_{b1} - V_{b2}) \ . \tag{21}$$

The errors stated above can be overcome by adjusting the bias voltages V_{b1}, V_{b2}. It was discussed in [5].

2.4. Noise margin

Noise margin is an important specific to a down literal circuit in MVL, as well as a CMOS inverter in binary logic. Low noise margin NM_L and high noise margin NM_H are defined as Eq.(22) and Eq.(23), respectively.

$$NM_L = V_{IL} - V_{OL} \ , \tag{22}$$

$$NM_H = V_{OH} - V_{IH} \ , \tag{23}$$

where the values of $V_{OL} = i$ and $V_{OH} = i+1$ in down literal function D_i. The definitions of V_{IL} and V_{IH} are shown in Fig.2(d). They can be solved as Eqs.(24)(25) by the same way as those in CMOS inverter. [4]

$$V_{IL1} = \frac{3V_{DD} + 3V^\star_{tp} + 5V^\star_{tn}}{8} \ , \tag{24}$$

$$V_{IH1} = \frac{5V_{DD} + 5V^\star_{tp} + 3V^\star_{tn}}{8} \ , \tag{25}$$

where $V^\star_{tp} = 2V_{tp} - V_{b1} + V_{DD}, V^\star_{tn} = 2V_{tn} - V_{b2}$ according to Eqs.(9)(10), and $w_{n1,2} = w_{p1,2} = w = \frac{1}{2}$.

The variable-threshold C-νMOS inverter presented by Shibata [3] has the same function as the νMOS down literal circuit proposed in this paper. However, their noise margins are different. V_{IL}, V_{IH} in the variable-threshold C-νMOS inverter are given by Eqs.(26)(27), respectively.

$$V_{IL2} = \frac{3V_{DD} + 3V_{tp} + 5V_{tn}}{4} - \frac{1}{2}(V_{b1} + V_{b2}) \ , \tag{26}$$

$$V_{IH2} = \frac{5V_{DD} + 5V_{tp} + 3V_{tn}}{4} - \frac{1}{2}(V_{b1} + V_{b2}) \ . \tag{27}$$

Then according to Eqs.(19-22), the differences of noise margins between the circuit of Shibata and the circuit proposed here can be given by Eqs.(28)(29).

$$\Delta NM_L = NM_{L1} - NM_{L2} = \frac{V_{b1} - V_{b2}}{8} \ , \tag{28}$$

$$\Delta NM_H = NM_{H1} - NM_{H2} = \frac{V_{b1} - V_{b2}}{8} \ . \tag{29}$$

Therefore, the νMOS down literal circuit has greater noise margin than the circuit proposed by Shibata [3] only if setting $V_{b1} > V_{b2}$.

Table 1. Quaternary down literal

$V_{DD} = 3V_d$	V_{TC}	$V_{b1} + V_{b2}$	V_{b1}	V_{b2}
$D_0(x)$	$0.5V_d$	$5V_d$	$3V_d$	$2V_d$
$D_1(x)$	$1.5V_d$	$3V_d$	$2V_d$	V_d
$D_2(x)$	$2.5V_d$	V_d	V_d	0

2.5. Quaternary νMOS down literal circuit

According to Eq.(15), the quaternary down literal circuit can be designed easily. There are three kinds of down literal functions in the quaternary system. $D_i(x)(i = 0, 1, 2)$ can be attained by setting V_{b1} and V_{b2} to the values stated in Table 1.

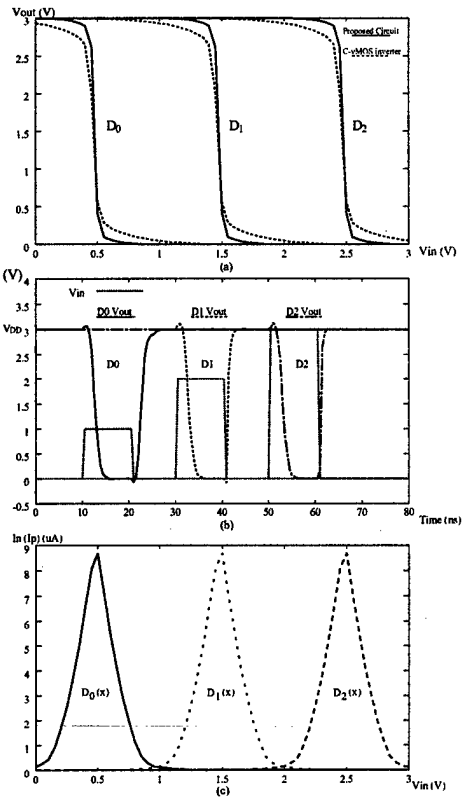


Figure 3. Simulation results of νMOS down literal circuit

Simulations of the quaternary νMOS down literal circuit were done using HSPICE with MOSIS 2.0μm CMOS device parameters. Under the conditions of V_d = 1V, W_n/L_n = 32μm/4μm, W_p/L_p = 80μm/4μm (to make $K_p \cong K_n$), $V_{tn} \cong |V_{tp}|$ = 1.0V, capacitances in νMOS 0.8pF, the DC analysis of $D_0(x), D_1(x), D_2(x)$ are shown in Fig.3(a). The gradient of V_{out} is about $V_{out}/Vin = 42.8$. The simulation

of Shibata's variable-threshold C-νMOS inverter was also done using the same device parameters as those of νMOS down literal circuit, and its transfer characteristic is also shown in Fig.3(a). The noise margin of νMOS down literal circuit is about 0.13V greater than that of variable-threshold C-νMOS inverter, which is accord with Eqs.(28)(29). If switching sensitivity is defined as $S = \frac{95\%V_{DD}}{\Delta V_{in}}, \Delta V_{in} = V_{in(V_{out}=97.5\%V_{DD})} - V_{in(V_{out}=2.5\%V_{DD})}$, there are $S = 12.95$ in the νMOS down literal circuit, and $S = 3.27$ in the variable-threshold C-νMOS inverter. The delay time of the proposed circuit is approximately 1.9 ns (Fig.3(b)). The peak currents of $D_0(x), D_1(x), D_2(x)$ are shown in Fig.3(c). They are very small with a value of 8.7μA.

3. Applications

The down literal function is one of the essential building block in MVL, and most MVL functions are combinations of kinds of down literal functions. In this section, we propose quaternary νMOS literal and T-gate circuits using the presented νMOS down literal circuits.

3.1 Quaternary νMOS literal circuits

The quaternary literal function is expressed by Eq.(30)[1]

$$x^{ab} = \begin{cases} R-1 & a \le x \le b \\ 0 & Others \end{cases}, \qquad (30)$$

where $x, a, b \in \{0,1,2,3\}$. The literal circuit can be composed of down literal circuits. The block diagram is shown in Fig.4. When the bias voltages of this circuit are set as given in Table 2 according to quaternary νMOS down literal circuit parameters, the quaternary literal functions: x^{11}, x^{12}, x^{22} can be realized. Their simulation results are shown in Fig.5. Simulation parameters are the same as those used in the quaternary νMOS down literal circuit.

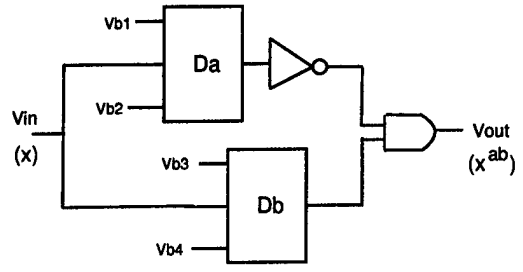


Figure 4. νMOS quaternary literal

Table 2. Quaternary literal

$V_{DD} = 3V_d$	V_{b1}	V_{b2}	V_{b3}	V_{b4}
x^{11}	$3V_d$	$2V_d$	$2V_d$	V_d
x^{12}	$3V_d$	$2V_d$	V_d	0
x^{22}	$2V_d$	V_d	V_d	0

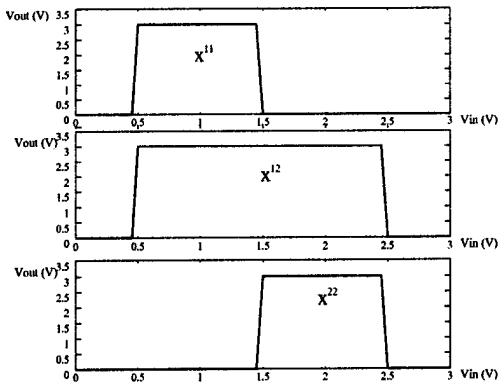


Figure 5. Simulation results of νMOS quaternary literal circuit

3.2 Quaternary νMOS T-gate circuit

The T-gate is a functional block in MVL. It is often applied in MVL image processing, such as quaternary pattern matching. There is a quaternary T-gate circuit with NMOS proposed in [1]. It requires multi-level ion implantation technology to set 5 different threshold voltages of enhancement and depletion MOS transistors, which increases the complexity and the fabrication cost.

Basing on quaternary νMOS down literal circuit and

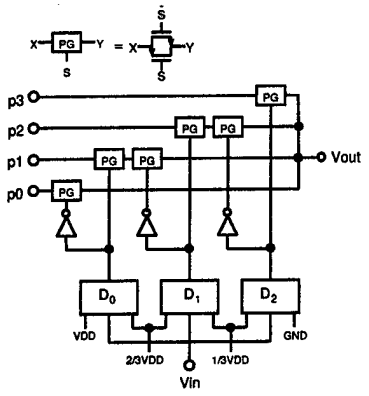


Figure 6. νMOS quaternary T-gate

CMOS pass gates, a νMOS quaternary T-gate circuit is built to complete the T-gate function denoted in Eq.(31).

$$T(p_0,p_1,p_2,p_3;x) = \begin{cases} p_0 & x=0 \\ p_1 & x=1 \\ p_2 & x=2 \\ p_3 & x=3 \end{cases} \quad (31)$$

where $p_0,p_1,p_2,p_3,x \in \{0,1,2,3\}$. The circuit is shown in Fig.6. The whole circuit can be fabricated only using a standard CMOS process. The performance of the proposed T-gate is validated by the simulation result of T-gate: $T(0,1,2,3;x)$ shown in Fig.7. The maximum delay time of this νMOS T-gate is 7.95ns. It is reduced about 12.5 times compared with that of multi-ion implantation involved CP-gate quantizer proposed in [6].

4 Conclusion

In this paper, the voltage-mode down literal circuit using two 2-gate νMOS transistors has been proposed. Compared to the conventional circuit, the νMOS down literal circuit has several advantages. It can be fabricated easily by general CMOS process, without multi-level ion implantation necessary in the conventional voltage-mode MVL circuits. It has the flexibility that one νMOS down literal circuit can be used to realize different down literal functions by setting bias voltages. The proposed down literal circuit has been verified by HSPICE simulations on the quaternary νMOS down literal circuit. It can work under 3V with 8.7μA peak consumption current and 1.9 ns delay time, and has greater noise margin and switching sensitivity than C-νMOS inverter proposed by Shibata. Basing on the quaternary νMOS down literal circuits, quaternary νMOS literal and T-gate circuits are also designed and proved by simulations.

With the proposed circuits, it is possible to fabricate various voltage-mode MVL circuits by the standard CMOS process, and reduce the cost. In addition, because νPMOS and νNMOS in the proposed down literal circuit have their own floating gates, compared with the C-νMOS inverter proposed by Shibata, multi-valued SRAM can be designed according to [7]. This work will be done in the further research.

Acknowledgments

The work reported in this paper is supported in part by the grant-in-aid for Scientific Research of the Ministry of Education, Science, Sports and Culture of Japan under Grant: 10650339.

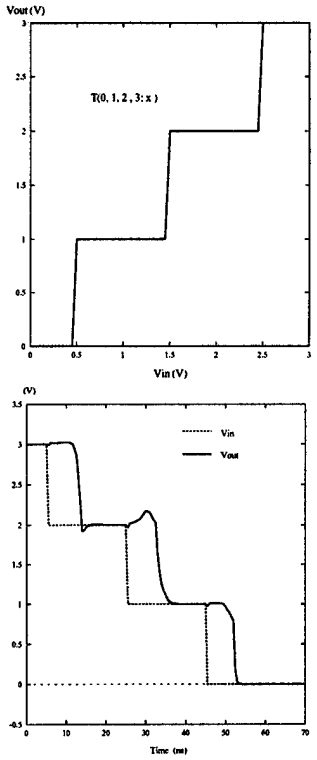


Figure 7. Simulation results of $T(0,1,2,3;x)$

References

[1] T. Higuchi and M. Kameyama. *Multiple-Valued Digital Processing System.* Syokodo, Japan, 1989.

[2] J. Shen, K. Tanno, O.Ishizuka, and Z. Tang. Application of neuron-MOS to current-mode multi-valued logic circuits. In *Proc.28th ISMVL*, pages 128–133, 1998.

[3] T.Shibata and T.Ohmi. A functional MOS transistor featuring gate-level weighted sum and threshold operations. *IEEE Trans. Electron Devices*, 39(6):1444–1455, 1992.

[4] N. H.E.Weste. *Principles of CMOS VLSI Design: A Systems Perspective.* AT&T Bell Laboratories, Incorporated, and Kamran Eshraghian, 1985.

[5] J. Shen, K. Tanno, and O.Ishizuka. Design and analysis of down literal circuit using neuron-MOS transistors. In *Proc.12th MVL Research Society of Japan*, pages 1–9, 1999.

[6] M. Kameyama, K. Horie, and T. Higuchi. Design of a vlsi-oriented quaternary complementary pass gate network. *IEICE*, J71-D(4):636–643, 1988.

[7] O.Ishizuka, Z. Tang, and H. Matsumoto. On design of multiple-valued static random-access-memsory. In *Proc. 20th ISMVL*, pages 11–17, 1990.

Arithmetic Circuits for Analog Digits

Aryan Saed*, Majid Ahmadi, Graham A. Jullien
*Nortel Networks, Nepean, Ontario, Canada
VLSI Research Group, University of Windsor
saed@nortelnetworks.com

Abstract

The Overlap Resolution Number System (ORNS) employs bit level analog residue arithmetic, and opens up a powerful approach to digital computing. This new redundant representation of signals, with Continuous Valued Digits, presents new methods for binary arithmetic and digital signal processing. The number system is based on analog residue digits, as opposed to binary or multiple-valued digit levels. Importantly, arithmetic in ORNS is tolerant to VLSI circuit tolerances. This allows simple elementary analog circuits to be employed, targeting digital accuracy.

1. Introduction

This paper presents analog CMOS circuit techniques for elementary ORNS arithmetic digit manipulations. The focus lies on binary multiplier structures that internally employ Continuous Valued Digits. The principles of arithmetic in the Overlap Resolution Number System differ significantly from the familiar concepts of arithmetic with binary or multiple-valued digits: logic gates, or respectively discrete level analog circuits, are replaced by continuous valued digit manipulation circuits.

Multiple-Valued Logic (MVL) circuit techniques exploit the potential accuracy of silicon circuits by relying on more than just the two states *on* and *off*. Modern circuits for MVL arithmetic units successfully employ analog circuits for elementary digit operations, in replacement of digital logic gates [1].

This paper discusses an alternative to binary and MVL multiplier structures. In the following section we review the new theory of Continuous Valued Digits (CVD's), and we compare their properties with the familiar discrete valued digits in a positional number system (PNS). We will see that already so called *binary* CVD's are capable of exploiting silicon accuracy in arithmetic structures, without resorting to higher radix values.

Arithmetic rules in ORNS are then reviewed, and their impact on hardware VLSI architectures is discussed with an emphasis on the implementation of binary digital multiplication functions. The paper finally proceeds with introduction of analog CMOS circuits for CVD's. These circuits comprise the leaf cells in hardware architectures for arithmetic building blocks.

2. Radix-B Overlap Resolution

Given a real x bound by X as $|x|<X$, we shall represent it by a set of CVD's r_n, with index $n=K\ldots L$ and $K\le L$. We recall from [2] and [3] that there exist two methods to calculate CVD's, each arriving at the same result. The first method involves a cascaded approach, whereby we start with the Most Significant Digit (MSD) r_L, and compute it as $r_L = B\cdot\frac{x}{X}$. The positive integer B is the radix, and we shall further assume $B\ge 2$. For $B=2$ we have the important case of binary ORNS. Further digits $n<L$ are calculated by the *cascade rule*:

$$r_n = (r_{n+1} - \tilde{a}_{n+1})\cdot B \qquad (1)$$

with $\tilde{a}_n = \lfloor r_n \rfloor$, whereby $\tilde{a}_n$ is an integer associated with the CVD r_n. The operator $\lfloor . \rfloor$ denotes flooring towards zero, such that $|\tilde{a}_n| \le |r_n|$. As a result we have $|r_n|<B$ and $|\tilde{a}_n| \le B-1$. The CVD's r_n need not be an integer. We choose to select L such that $B^{L+1} \ge |X|$. A rule for selecting K will follow in the next section.

The second method involves the signed modulo operation $a \bmod B = a - B\lfloor a/B \rfloor$, which we define for

0-7695-0161-3/99 $10.00 © 1999 IEEE

Table 1: ORNS Example for x=58.742

	Decimal ($B=10$)			Binary ($B=2$)		
n	r_n	$\tilde{a}_n$	$q_n/(\mu A)$	r_n	$\tilde{a}_n$	$q_n/(\mu A)$
6	-		-	1.17484	1	29.371
5	-		-	0.34968	0	8.742
4	-		-	0.69936	0	17.484
3	-		-	1.39672	1	34.918
2	0.58742	0	2.9371	0.79744	0	19.936
1	5.87420	5	29.3710	1.59488	1	39.872
0	8.74200	8	43.7100	1.18796	1	29.699
-1	7.42000	7	37.1000	0.37952	0	9.488

integer as well as real values of a for both signs. The ORNS *basic expression* is:

$$r_n = \left(\frac{x}{X}\cdot B^{L-n+1}\right) \mathrm{mod} B \qquad (2)$$

Both methods may also be used to compute digits with index $n>L$. We shall see in following sections that such *excessively evolved digits* (EED's) serve arithmetic with CVD's. EED's are equivalently computed by $r_{n>L} = r_{n-1}/B$ conform the cascade rule, or by $r_{n\geq L} = B^{L-n+1}\cdot x/X$ conform the basic expression. The proof is simple, and follows from properties of the modulo operation. We conclude that CVD's in ORNS are of the general form $(aB^{-k})\mathrm{mod}B$, with real a.

An ORNS number is written as $N_x = (r_L,\ldots,r_0|r_{-1},\ldots r_K)$, with a radix 'bar' between r_0 and r_{-1}. We typically use a decimal notation for the value of a CVD, and hence a vertical bar shall be used for the radix point of the ORNS number. In this paper we shall limit our discussions to non-negative values of x.

A remarkable characteristic of a CVD is, as the name implies, that a digit value requires a form of continuous symbolization. Hence, we may also term CVD's as *analog digits*, if we have an electronic implementation in mind. If a linear electronic medium of our choice, for instance a current, charge or voltage, ranges from 0 to $\pm Q$ units, then each CVD is matched proportionately to an electronic quantity q_n by $q_n = r_n\cdot Q/B$.

Example 1: A value $x\geq 0$, limited by $X = 100$, shall be represented by CVD's in the range $0\mu A$ to $50\mu A$. We select two radix values, $B = 10$ for *decimal* ORNS, and $B = 2$ for *binary* ORNS. To satisfy $X\leq B^{L+1}$ we select $L = 1$ for the decimal case, and $L = 6$ for the binary case. For both we select $K = -1$. The CVD's for $x = 58.742$ are presented in Table 1. The decimal digit r_2 is an EED. We observe, that $\tilde{a}_n$ are the PNS digits of $(xB^{L+1})/X$. Notice, that the term *binary* refers to the radix $B = 2$, while the digits r_n are not at all binary valued.

3. Redundancy

The value x shall be termed the *root* of the number N_x and it is retrieved from N_x by the MSD alone, without any error: $x = r_L\cdot X/B$. N_x is not an approximate representation, since r_L and x may assume any real value. However, there is a practical limitation. With an increasing precision of x, it becomes difficult, if not eventually impossible, to maintain that precision in r_L when a circuit implementation with q_L is envisioned. We therefore need to discuss the consequences of limited precision digits in N_x.

In a positional number system the integers $\tilde{a}_n$ are stored as multiple-valued digits, and arithmetic is performed with these discrete valued digits, ranging $0\ldots(B-1)$ for $x\geq 0$.

Table 2: DVD and CVD Error Threshold Examples

Radix B	2	4	8	10	100
$\hat{\vartheta}$ DVD	50.0%	16.7%	7.14%	5.56%	0.51%
$\hat{\vartheta}$ CVD	16.7%	10.0%	5.56%	4.55%	0.50%

Given two neighbouring digits r_n and r_{n-1}, we are able to retrieve $\tilde{a}_n$ by $\tilde{a}_n = \lfloor r_n \rfloor$, and recalculate r_n as $r_n = \tilde{a}_n + r_{n-1}/B$. This is a trivial result from (1), and equally trivial we can retrieve $\tilde{a}_n$ by $\tilde{a}_n = r_n - r_{n-1}/B$. Given an errored digit pair r'_n and r'_{n+1}, we restore r'_n to $r''_n = \tilde{a}'_n + r'_{n-1}/B$, whereby we calculate the associated integer as $\tilde{a}'_n = [r'_n - r'_{n-1}/B]$, whereby [.] denotes rounding to the nearest integer. With $r'_n = r_n + \varepsilon_n$ and $r'_{n-1} = r_{n-1} + \varepsilon_{n-1}$, we find $\tilde{a}'_n = [\tilde{a}_n + \varepsilon_n - \varepsilon_{n-1}/B]$. Hence, under the condition

$$|\varepsilon_n - \varepsilon_{n-1}/B| < 1/2 \tag{3}$$

we have $\tilde{a}'_n = \tilde{a}_n$. Provided that $\tilde{a}'_n = \tilde{a}_n$, we find that $r''_n = r_n + \varepsilon'_n$ contains the error $\varepsilon'_n = \varepsilon_{n-1}/B$. The success of obtaining $\tilde{a}'_n = \tilde{a}_n$ conveniently does not depend on the value of r_n. We show in [4] that the condition in (3) implies

$$|\varepsilon_n| < B/(2(B+1)) \tag{4}$$

if we assume equal error statistics among all digits. In comparison, the error condition for Multiple-Valued digits depends on the digit level distances, which equals one regardless of the radix. Hence, the error of discrete valued digits (DVD's) must adhere to $|\varepsilon_n| < 1/2$.

The important *relative* implementation error condition for CVD's and DVD's shall be defined as $|\varepsilon_n|/B < \hat{\vartheta}$ and various values of $\hat{\vartheta}$ are reported in Table 2. We recognize the familiar noise margin of 50% for binary systems, and clearly that noise tolerance is inferior for CVD's. It is therefore not particularly envisioned that ORNS be employed for storage of x. The important gain of CVD's results from arithmetic properties which we shall now discuss.

4. Addition and Multiplication

The CVD's $r_n(x+y)$ of a sum $x+y$ are simply obtained digit-wise and there is no carry or other interaction required between neighbouring digits, [3], [6]:

$$r_n(x+y) = (r_n(x) + r_n(y)) \bmod B \tag{5}$$

We recall that we have limited our discussion to $x, y \geq 0$. The CVD's of a product $\lambda \cdot x$ with integer λ are:

$$r_n(\lambda x) = (\lambda \cdot r_n(x)) \bmod B \tag{6}$$

In the special case $\lambda = B^k$, we have $r_n(B^k \cdot x) = r_{n-k}(x)$, and if $y = \sum_{\forall k} \lambda_k B^k$, then

$$r_n(y \cdot x) = \left(\sum_{\forall k} \lambda_k \cdot r_{n-k}(x) \right) \bmod B \tag{7}$$

This is the equivalent of the familiar shift-and-add principle for Radix-B multiplication. In binary ORNS the values of λ_k are limited to $\{0, 1\}$, and hence they serve as an on-off switch for summing analog voltages, currents or charges $r_{n-k}(x)$. We intend to represent the multiplier y in binary, with digits λ_k.

The rules for addition and multiplication not only hold for the perfect CVD's r_n, but also for imperfect values r'_n. Of course, CVD imperfections of operands in addition and multiplication propagate into the result CVD's, but it has been demonstrated here, as well as in [2] and [4], that ORNS is robust against such errors.

Example 2: Given $x = 31.89$ and $y = 3.54$, they shall be summed in decimal ORNS with $X = 100$. Their ORNS numbers with approximate CVD's are $N'_x = (3.2, 1.9 | 8.9)$ and $N'_y = (0.4, 3.5 | 5.4)$. The perfect sum of errored digits is $(3.6, 5.4 | 4.3)$, yet we shall consider the *errored* sum with *errored* digits

$N'_{x+y} = (3.61, 5.39|4.31)$. We find $\tilde{a}'_0 = [5.39 - 4.31/10] = 5$ and $r''_0 = 5 + 4.31/10 = 5.431$, and further $\tilde{a}'_1 = 3$ and $r''_1 = 3.5431$. We conclude, with a minor error, that $x+y = 35.431$. The error is reduced by increasing the number of digits.

5. ORNS-Digital Interface

In [5] we presented a method for generating CVD's directly from binary bits, and we discussed an ORNS based architecture for a binary multiplier. In the remaining sections of this paper we will employ our combined knowledge of the rule for addition and the error tolerance of CVD's, to develop CMOS current-mode analog circuits for Radix-2 (binary) ORNS.

Considering the novelty of this topic, we will review the generation of CVD's from bits, and the retrieval of bits from CVD's. This will allow us to use the arithmetic properties of CVD's within a binary multiplier. This encapsulation of ORNS within a digital cast is one application of the number system. It allows the development of arithmetic standard cells in a digital circuit library.

A second application of ORNS lies in the immediate representation of a signal value x by CVD's, without the intervention of a binary number system. In essence this is an alternative to quantization and digitalization of a signal, whereby we are still able to effectively distribute x over a set of digits. In this paper are concerned with the first application, in particular the development of a binary multiplier.

A number x is represented by a weighted sum over binary bits a_n

$$x = \sum_{n=K'}^{L'} a_n \cdot 2^n \tag{8}$$

Considering the range of x in Eqn. (8), we select $X = 2^{L'+1} = 2^{L+1}$. We require $L' \geq K'$, but allow $L', K' \leq 0$. Inserting Eqn. (8) in Eqn. (2), and selecting $B = 2$ we find the CVD's of x directly from a weighted sum of the bits:

$$r_{L-j} = \sum_{n=K'}^{L'-j} a_n \cdot 2^{n+j-L'} \tag{9}$$

With a modification of the radix in Eqn. (8) we can obtain CVD's from non-binary PNS digits by selecting B accordingly. We shall exploit the error tolerance of Eqn. (4), and limit the scope of the summation in Eqn. (9) to only a few (ψ) bits per CVD:

$$r'_{L-j} = \sum_{n=L'-j-\psi+1}^{L'-j} a_n \cdot 2^{n+j-L'} \tag{10}$$

The relative digit error $\hat{\varepsilon}_n = \varepsilon_n/2$ results from the truncation, and it is bound by $|\hat{\varepsilon}_n| < 2^{-\psi}$. The bits a_n of x equal the associated integers $\tilde{a}_n$. A typical value of $\psi = 4$ implies a 4-bit digital-to-analog (DA) conversion per CVD, regardless of the binary word length of x. Bits are calculated from CVD's by

$$\tilde{a}'_n = \left[r'_n - \frac{r''_{n-1}}{B} \right] \bmod^+ B \tag{11}$$

We define $a \bmod^+ B = a + I' \cdot B$ with integer I' such that $0 \leq a + I' \cdot B < B$ for real a. Digit r'_n is now restored by

$$r''_n = \left(\frac{r''_{n-1}}{B} + \tilde{a}'_n \right) \bmod^+ B \tag{12}$$

We obtain $\tilde{a}'_n$ and r''_n from least to most significant, in sequence, utilising the *corrected* neighbour r''_{n-1}. Of course, the least significant digit remains uncorrected: $r''_K = r'_K$.

6. Binary Multiplication

In Figure 1 we present a 4bit-by-4bit multiplication example. It is noted that the applied word-length of 4 bits is not related to $\psi = 4$. There are seven DA converters in the top, producing digits $0 \leq r_n < 2$. The four rows of switches are driven by λ_k's. The summers at the bottom are 4-input modulo summers, and the triangles are symbols for the digit correction and bit retrieval operations of Eqn. (11) and Eqn. (12). Digit r'_n

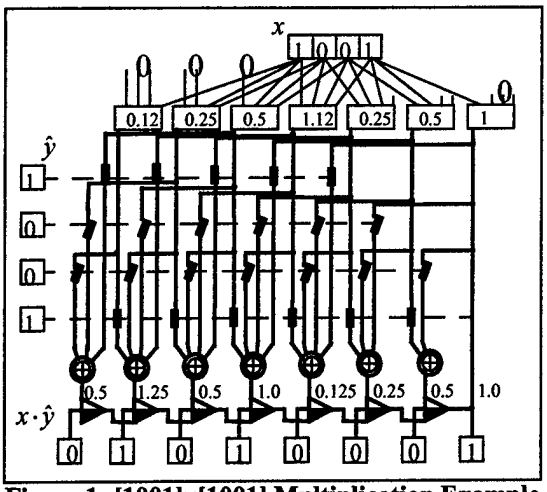


Figure 1: [1001]x[1001] Multiplication Example

is entered at the top, and r''_{n-1} at the right. The output r''_n is obtained from the bottom, and $\tilde{a}'_n$ from the left.

The size of the multiplier has been selected such, that it illustrates the principles of the architecture. The number of columns can easily be extended to ν bits, with typical values of 16, 32 and 64. A ν-by-ν multiplier of such dimensions will consist of μ-bit layers, each ν wide. The value of μ is determined by the analog accuracy of the employed circuits. In our example we employed 4-bit analog accuracy ($\mu=4$), and hence a 32-by-32 bit multiplier consists of eight 4-bit layers.

Layers of switches are separated by layers of summers and correction units. The latter are depicted by triangles in Figure 1, and in essence they refresh the digit values. The DA converters are only needed once. Their word-length is determined by the targeted analog accuracy. It is only required to provide a DA resolution that is intended to be supported by the accuracy of the applied circuits.

7. CMOS Current-Mode Analog Circuits

We will now introduce two characteristic circuits for radix-2 (binary ORNS) analog digits: the digit correction circuit that performs the operations of Eqn. (11) and Eqn. (12), and a 2-input modulo summer, which is a building block for a 4-input summer. In Figure 2 we present the two circuits. Of course, eventually the 4-input summer is best designed directly.

The left circuit performs the rule for addition in Eqn. (5). Since the sum within the modulo operation is limited to $0 \le (r_n(x)+r_n(y)) < 2B$, we can write

$$r_n(x+y) = r_n(x)+r_n(y)-\kappa B \qquad (13)$$

whereby $\kappa \in \{0,1\}$. The draining currents *I0* and *I1* represent the summands $r_n(x)$ and $r_n(y)$, and the modulo sum $r_n(x+y)$ is drained as *I2*. The voltage at node *C* represents κ. The sum $r_n(x)+r_n(y)$ at *M4* is compared with a reference current at *M2*, representing the digit value B. We recall that Q corresponds to B.

The comparison results in 'greater' or 'less', leading to *C* 'low' or 'high' respectively. In the case of 'less',

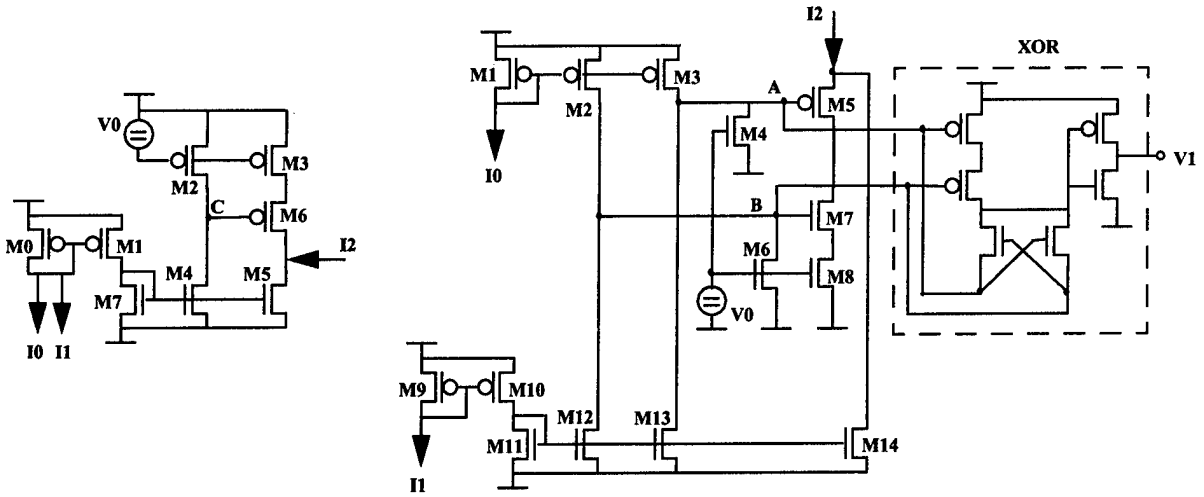


Figure 2: CMOS analog current-mode circuits for a CVD full adder (left) and CVD correction with binary conversion (right)

the sum is simply presented at the output by *M5* ($\kappa = 0$). In the case of 'greater', *M6* is turned on, and the value B at *M3* is subtracted from the sum ($\kappa = 1$).

The right circuit in Figure 2 performs the digit correction and delivers a binary bit for each CVD. Digit r'_n is drained as *I0*, and r''_{n-1} as *I1*. The output r''_n is drained as *I2*. The corresponding bit $\tilde{a}'_n$ is provided as a voltage *V1*.

The modulo rounding operation in Eqn. (11) is performed by comparing the difference $r'_n - r''_{n-1}/B$ to the digit values 1/2 and 2/3. These values correspond to quantities $Q/4$ and $3Q/4$. If the difference lies between the two, we conclude $\tilde{a}'_n = 1$, otherwise $\tilde{a}'_n = 0$. The current through *M4* equals $Q/4$, and the current through *M6* equals $3Q/4$. The comparisons are separately evaluated at nodes *A* and *B*, and the *XOR* delivers the bit as a voltage *V1*. In accordance with Eqn. (12), for $\tilde{a}'_n = 1$ the cascade of switches *M5* and *M7* adds a current of $Q/2$ at *M18* to r''_{n-1}/B at *M14*.

The presented circuits have been designed and simulated in a 0.8μm CMOS environment, with 3.3V supply voltage and $Q = 50\mu A$. The 4-by-4 multiplier of Figure 1 has been successfully designed, based on these circuits. The optimization for speed remains a topic of future research.

The reference voltage *V0* in both circuits may be fixed, or adjusted to compensate for temperature variations. The accuracy of the current mirrors chiefly determines the quality of the circuit. With increased accuracy the accumulated digit error is reduced, and the column size in a multiplier may be increased, resulting in fewer layers. Simpler circuits with smaller transistors introduce more error, and hence the architectural complexity increases. The design of an ORNS based arithmetic library cell is therefore directed by such trade-offs, whereby speed, power consumption, area, and switching noise are components in the cost function.

8. Conclusions

This paper has discussed the role of analog digits in the Overlap Resolution Number System, and it has reviewed arithmetic rules and interfaces between multiple-valued digits and continuous valued digits, with a particular focus on the binary case. We have learned that a binary multiplier of any dimension can be assembled from layers of switches and digit refreshment circuits. Analog circuit tolerance is not a prohibitive limitation for large multipliers.

ORNS exploits VLSI circuit accuracy without resorting to a radix higher than $B = 2$. Nevertheless, the presented theories allow for instance quaternary ORNS ($B = 4$) to interface with 4-level MVL.

CMOS analog current mode circuits for ORNS are closely related to current mode circuits in MVL. They consist of familiar structures of current comparators and current mirrors.

9. References

[1] Butler, Jon T. ed., Multiple-Valued logic in VLSI, IEEE Computer Society Press, 1991

[2] Saed, A., M. Ahmadi, G.A. Jullien, W.C. Miller, "Overlap Resolution: Continuous Valued Digits for Hybrid Architectures", 40th Midwest Symposium on Circuits and Systems, August 1997, Sacramento, California.

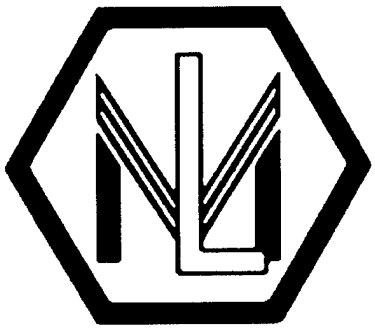
[3] Saed, A., M. Ahmadi, G.A. Jullien, W.C. Miller, "Overlap Resolution: Arithmetic with Continuous Valued Digits in Hybrid Architectures", Thirty First Annual Asilomar Conference on Signals, Systems, and Computers, November 1997, Pacific Grove, California.

[4] Saed, A., M. Ahmadi, G.A. Jullien, W.C. Miller, "Circuit Tolerances and Word Lengths in Overlap Resolution", The 1998 IEEE International Symposium on Circuits and Systems (ISCAS), June 1998, Monterey, California.

[5] Saed, A., M. Ahmadi, G.A. Jullien, "Analog Digits: Bit Level Redundancy in a Binary Multiplier", Thirty Second Annual Asilomar Conference on Signals, Systems, and Computers, November 1998, Pacific Grove, California.

[6] Saed, A., M. Ahmadi, G.A. Jullien, "Arithmetic with Signed Analog Digits", 14th IEEE Symposium on Computer Arithmetic, April 1999, Adelaide Australia.

SESSION VIIA
APPLICATIONS
CHAIR: Radomir Stanković



Quaternary Coded Genetic Algorithms

Kai Freitag
SAG Systemhaus GmbH
Elsenheimer Str. 11, 80687 Munich
Germany

Lars Hildebrand, Claudio Moraga
Dept. Computer Science and Computer Engineering
University of Dortmund; 44221 Dortmund
Germany

Abstract

Genetic Algorithms comprise search and optimization strategies which are inspired by natural evolution: "survival of the fittest". In most of the best known basic genetic algorithms a binary coding of solution candidates is used. However for the DNA-coding, mother nature uses 4 purine bases: adenine (A), cytosine (C), guanine (G) and thymine (T). Following this idea, the present paper studies quaternary coded genetic algorithms and, based on high complex test functions, shows that these algorithms have a performance as good as their corresponding binary versions for problems with low dimensions and reach very fast an acceptable good fitness, if not the best, for high dimensional problems. For the first time it is shown, that the performance of genetic algorithms under a Gray code is sensitive to permutation of the columns of the code.

1. Introduction

Genetic Algorithms were introduced by John Holland over two decades ago [Holl 75] and have been intensively studied ever since (see *e.g.* [Gold 89], [Tane 89], [Davi 91], [Eshe 91], [Mühl 91], [ScEs 93]).

Following the original idea of J. Holland most genetic algorithms use a binary coding of their individuals. Real-coded genetic algorithms have been developed and used with success (see *e.g.* [Eshe 91], [HeLM 98]) thus becoming very close to evolutionary algorithms [Rech 73], [Schw 75] which were developed for real parameter optimization and consequently work always with real-valued individuals.

One of the particularly most representative codings found in the case of human genetics is that of DNA chains, which uses only four purine bases (A, C, G and T) between a two–stranded winding chain of deoxyribose and phosphate, thus suggesting, that for an analog purpose, nature prefers a quaternary code. This observation motivated the present comparative study of binary and quaternary coded genetic algorithms. Nevertheless, it has been argued that by using higher cardinality coding alphabets, fewer hyperplane partitions would be available and larger populations would be necessary (see *e.g.* [Whit 93]). The results of the present paper show that this is not the case.

The paper is structured as follows. In the next section, the basic principles of genetic algorithms will be explained. This will be followed by the presentation of the different aspects to be considered when moving from a binary to a quaternary coding. A special section will be devoted to introduce the test functions used to evaluate the performance of the algorithms. The paper will be closed with a careful analysis of the test-results.

2. The basic genetic algorithm

Genetic Algorithms are search and optimization procedures inspired by biological genetics (selection, reproduction, mutation) which exhibit a high degree of robustness and have proven to offer good solutions to a number of relevant hard to solve problems.

The implementation of a genetic algorithm comprises the following main steps:

1. Random initialization of a properly coded population of individuals that represent candidates for the solution of the given problem.
2. Evaluation of the goodness of each individual in the population. (Fitness function)
3. Selection of the best individuals with a probability proportional to their relative fitness. Production of an intermediate new population by randomly choosing mating pairs of selected individuals and applying genetic operations: crossover (exchange of a randomly selected substring between two individuals) and mutation (change in the value of a randomly chosen coding position).
4. Evaluation of the individuals of the intermediate population.
5. Termination of the algorithm if a computing time limit or a satisfactory solution has been reached. (Output: the best individual). Otherwise continuation at step 3.

Quaternary coded genetic algorithms

It becomes apparent that the structure of a genetic algorithm will basically not change if a quaternary encod ing is used. In the case of binary coded genetic algorithms

0-7695-0161-3/99 $10.00 © 1999 IEEE

the most frequently used encodings are the position-weighted and the Gray code.

It will be first analyzed whether besides the natural quaternary coding, obtained from the coefficients of a polynomial representation of integers by using powers of 4, a quaternary Gray code may be defined. This question may be positively answered. A *scheme* to generate a quaternary Gray code is shown in figure 1, where the "fingers" have to be properly enlarged in the case of a higher number of variables. It becomes apparent that it is always possible to construct in this way a Hamilton-path in $(Z_4)^n$. Moreover it is fairly obvious that if the columns of a Gray Code are permuted, the resulting Code is also Gray.

With respect to genetic operations, both for crossover and selection the corresponding operations of the binary case may be applied. For the mutation however, the quaternary case offers more possibilities. Three mutation operations will be discussed below:

Let $(a_{n-1}, ..., a_1, a_0) \in (Z_4)^n$ represent the coding of an individual and let a'_i denote a mutated value at the i-th position. The following mutation strategies will be considered:

1. Random mutation. For every position $0 \le i \le n-1$ and taking in account the predefined mutation probability, "flip a coin" to decide whether the value a_i should be mutated or not. If yes, generate a random number $r \in Z_4$ and let $a'_i = r$. Notice that if $r = a_i$, there will obviously be no change. This is allowed.

2. Inversion mutation. Define L to be one fourth of the length n of an individual. Considering the mutation probability, decide whether an individual should be mutated or not. If yes, generate a random number $\lambda \in N$, $L \ge \lambda > 0$, s.t. its probability distribution is inverse proportional to the number of generations. Moreover choose randomly the mutation start position s, where $1 \le s \le n-\lambda$. Finally let $a'_{s+i} = a_{s+\lambda-i}$ for all i s.t. $0 \le i \le \lambda$, *i.e.*starting at position s a substring of length λ will be replaced by its mirrored image. (NB: This mutation type was also used in the binary case.)

3. Delta mutation. This mutation type is based on the non-uniform mutation introduced by Z. Michalewicz [Mich 92]. For every position $0 \le i \le n-1$ and taking in account the predefined mutation probability, "flip a coin" to decide whether the value a_i should be mutated or not. If yes, generate a random bit $b \in \{0, 1\}$ and let

$$a'_i = b[a_i + \Delta(g, 3-a_i)] + (1-b)[a_i - \Delta(g, a_i)]$$

where g denotes the number of the present generation (the initial population represents the 0-th generation), and Δ is defined as

$$\Delta : N_0 \times \{0, 1, 2, 3\} \rightarrow \{0, 1, 2, 3\}$$
$$\Delta(g, a) = a^* \text{ with } a^* \in \{0, ..., a\}$$

where the probability of producing smaller values increases with g.

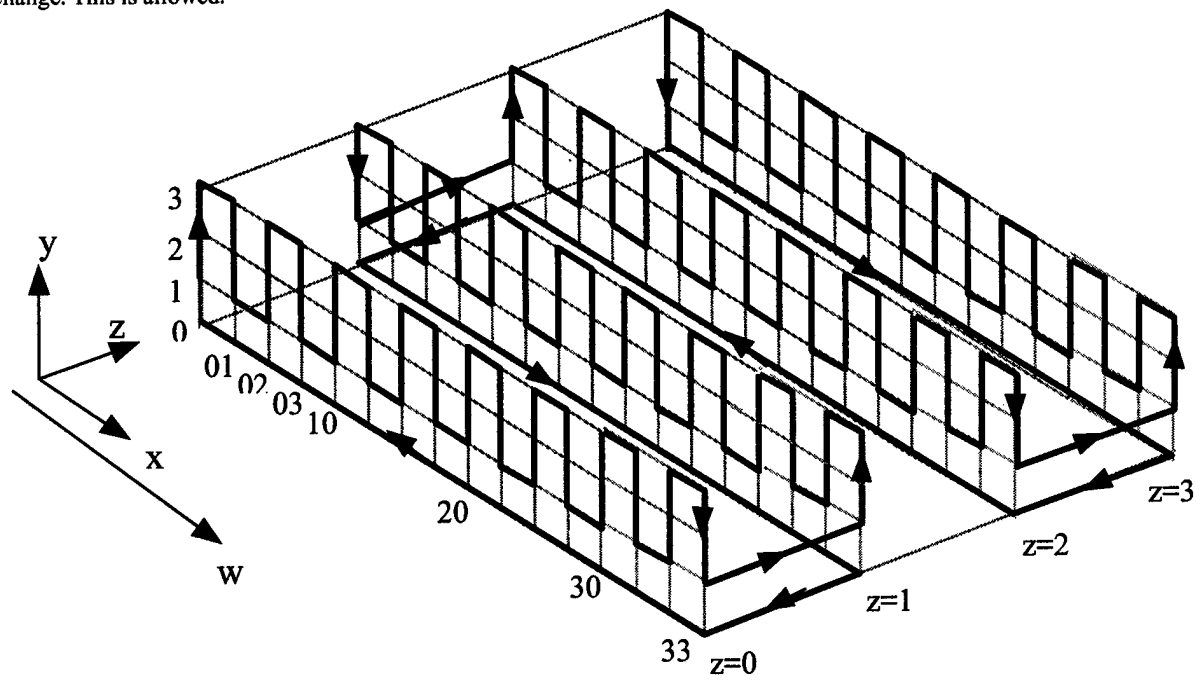


Fig. 1: Scheme for the generation of quaternary Gray codes

3. The choice of test functions and algorithm parameters

In order to evaluate the performance of genetic algorithms, several special test functions have been proposed. For this paper, the test functions suggested in [WRDM 96] have been used, since they have been especially developed to avoid deficiencies of former test functions. Moreover a method to gradually increase their complexity has been disclosed in [WRDM 96].

The functions are:

$$F1(x,y) = -x \sin\sqrt{|x-(y+47)|} - (y+47)\sin\sqrt{\left|y+47+\frac{x}{2}\right|}$$

$$F2(x,y) = x \sin\left(\sqrt{|y+1-x|}\right)\cos\sqrt{|y+1+x|} + $$

$$+ (y+1)\cos\left(\sqrt{|y+1-x|}\right)\sin\sqrt{|y+1+x|}$$

with $x,y \in [-512 ; 511]$

$$F3(x,y) = 0.5 + \frac{\sin^2\sqrt{100\,x^2 + y^2x^2} - 0.5}{(1 + 0.001(x-y)^2)^2}$$

with $x,y \in [-100 ; 100]$

The following method to gradually increase the complexity of the test functions was used:

$$EF\alpha(x_1,\ldots,x_N) = \sum_{i=1}^{N-1} F\alpha(x_i, x_{i+1}) + F\alpha(x_N, x_1)$$

where $\alpha \in \{1, 2, 3\}$ and EF stands for "extended function". The notation $EF\alpha_N$ will be used to denote an extended α-function with N 2-place α-functions as components.

In order to adjust the basic parameters of the genetic algorithms to be tested (population size, probability of crossover and mutation, type of crossover, mutation and selection) a meta-genetic algorithm was used [Gref 86] upon the test function $EF1_3$ –(maximum search)- leading to the following results with respect to the Online–Performance (average of the fitness of all individuals by increasing number of generations) as shown in Table 1 and Offline–Performance (average of the best individual–fitness through the past generations) as shown in Table 2.

When adjusting the Parameters of the Online–Performance a 2–point–crossover and a tournament selection with elitism (except for Gray–4) was used. In the case of the Offline–Performance a 1–point–crossover (except for the quaternary Code: 2–p–c) and tournament selection with elitism was used [Gold 89]. It may be observed that the obtained population size is much larger than otherwise suggested in the literature (around 60); but this very much depends on the degree of difficulty posed by the function ($EF1_3$) selected to support the adjustment. More important is to notice that the population size for *both* binary and quaternary coded algorithms is very similar. It should also be noticed that for the Online–Performance the inversion mutation gave the best results, whereas this type of mutation is hardly mentioned in the usual literature.

Table1: Parameters for Online–Performance

Code	Pop. Size	$p_{mutation}$	$p_{cross.}$	Mutation type
Binary	380	0.005	0.50	Invers.
Gray–2	360	0.02	0.55	Invers.
Quatern.	410	0.001	0.65	Invers.
Gray–4	370	0.02	0.45	Invers.

Table 2: Parameters for the Offline–Performance

Code	Pop. size	$p_{mutation}$	$p_{cross.}$	Mutation type
Binary	390	0.090	0.65	Random
Gray–2	410	0.045	0.85	Compl.
Quatern.	380	0.150	1.00	Delta
Gray–4	400	0.150	0.55	Delta

4. Experimental results

The most important results of this study may be summarized as follows:

(i) In the case of problems with a relatively low complexity, the offline–performance of binary as well as quaternary coded algorithms are very similar, as shown in figure 2 with the test function $EF1_3$.

(ii) The pre–adjustment of the algorithm parameters by means of a meta genetic algorithm contributes to a noticeable improvement in performance, as may be seen in figure 3, where the offline– performance of optimized binary and quaternary algorithms processing $EF3_5$ are compared with results obtained with (binary coded) algorithms using the set of parameters suggested by K. DeJong [DeJo 75] and J. Grefenstette [Gref 86]. It should be noticed that the quaternary algorithms achieve their highest fitness with only 50 generations, meanwhile the straight binary needs over 600 generations. The Gray coded binary algorithm needs slightly over 300 generations to reach the same level of performance, but reaches later a roughly 5% better performance.

(iii) Figure 4 shows the result of processing $EF1_{10}$. It is simple to see that the quaternary algorithm reaches the

highest fitness much faster than the binary and binary Gray genetic algorithms.

(iv) The performance of the quaternary algorithm depends on the kind of mutation that is used. This is clearly disclosed in figure 5, where the processing of the test function $EF1_5$ is illustrated. Depending on the kind of mutation used, a difference in performance of up to 10% may be observed.

(v) A particularly important result is illustrated in figure 6, where it is shown that (at least in the case of the test function $EF3_5$) new quaternary Gray codes, obtained by simply permuting the columns of the original one exhibit a different performance. This suggests that one additional goal of the meta genetic algorithm should be the search for the best quaternary Gray code. This alone is a hard problem, since the search space is huge: every single Gray code leads to $(4^n)(n!)$ possible variations: there are 4^n different alternatives to code Zero and there are n! different permutations of the columns of the code.

5. Conclusions

Quaternary genetic algorithms have been introduced and a comparative analysis with binary algorithms has been done. Particularly difficult test functions were selected for this study in order to have results that are significant for real-world-problems. It has been shown that the optimal population size for both kinds of algorithms is about the same and that for functions of low dimension, binary and quaternary algorithms have roughly the same performance. In the case of higher dimensional functions, quaternary algorithms reach a high level of fitness much faster than the corresponding binary, even though binary algorithms will reach at the end a slightly better performance. In the case of quaternary algorithms there are different kinds of mutation and the choice of mutation affects the performance of the algorithm. Additionally, in the case of quaternary Gray coded algorithms, the choice of the particular Gray code also affects the performance of the algorithm. This is a quite new result. It is *conjectured* that these different Gray Codes drive the exploration and exploitation activities of the genetic algorithm into different regions of the problem space thus affecting the performance.

The results of this study suggest that in the case of high dimensional problems it might be convenient to start with a quaternary algorithm and by the time the fitness remains asymptotic stable, change to a binary coded one.

6. References

[Davi 91] Davis L.: *Handbuch of Genetic Algorithms.* Van Nostrand Reinhold, N.Y., (1991)

[DeJo 75] DeJong K.: Analysis of the behaviour of a class of genetic adaptove systems. Ph.D. Thesis, University of Michigan, (1975)

[Eshe 91] Eshelman L.: The CHC Adaptive Search Algorithm. In: *Foundations of Genetic Algorithms* (G. Rawlings, Ed.), Morgan Kaufmann, Los Altos CA., (1991)

[Frei 98] Freitag K.: Bewertung vierwertig kodierter genetischer Algorithmen. Master Thesis, Dept. of Computer Science and Computer Engineering, Univ. of Dortmund, Germany, (1998)

[Gold 89] Goldberg D.E.: *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison Wesley, Reading MA., (1989)

[Gref 86] Greffenstette J.J.: Optimization of control parameters for Genetic Algorithms. *IEEE Trans. on Systems, Man and Cybernetics* **16**, (1), 122–128, (1986)

[HeLM 98] Herrera F., Lozano M., Moraga C.: Hierarchical distributed GA. *Proc. Parallel Problems Solving from Nature*, Amsterdam, (1998)

[Holl 75] Holland J.: *Adaptation in Natural and Artificial Systems.* Univ. Michigan Press, (1975)

[Mich 92] Michalewicz Z.: *Genetic Algorithms + Data Structures = Evolution Programs.* Springer Verlag, Berlin, (1992)

[Mühl 91] Mühlenbein H.: Evolution in time and space: the parallel genetic algorithm. In: *Foundations of Genetic Algorithms* (G. Rawlings, Ed.), Morgan Kaufmann, Los Altos CA., (1991)

[Rech 73] Rechenberg I.: *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Friedrich Fromman Verlag, Stuttgart. (1973)

[ScEs 93] Schaffer J.D., Eshelman L.: Real–coded Genetic Algorithms and Interval Schemata. In: *Foundations of Genetic Algorithms* **2**, (D. Whitley, Ed.), Morgan Kaufmann, Los Altos CA., (1993)

[Schw 75] Schwefel H.P.: *Evolutionsstrategie und numerische Optimierung.* Dissertation, Technische Universität Berlin. (1975)

[Tane 89] Tanese R.: Distributed Genetic Algorithms. *Proc. 3rd. Int. Conf. on Genetic Algorithms.* Morgan Kaufmann, Los Altos CA., (1989)

[Whit 93] Whitley D.: A genetic Algorithm Tutorial. Technical Report CS–93–103, Dept. Comp. Science, Colorado State University, (1993)

[WRDM 96] Whitley D., S. Rana, J. Dzuberra and K.E. Mathias: Evaluating Evolutionary Algorithms. *Artificial Intelligence* **85**, 245–276, (1996)

Note: In all following pictures, the vertical axis corresponds to the relative fitness (function maximum) meanwhile the number of generations appears in the horizontal axis.

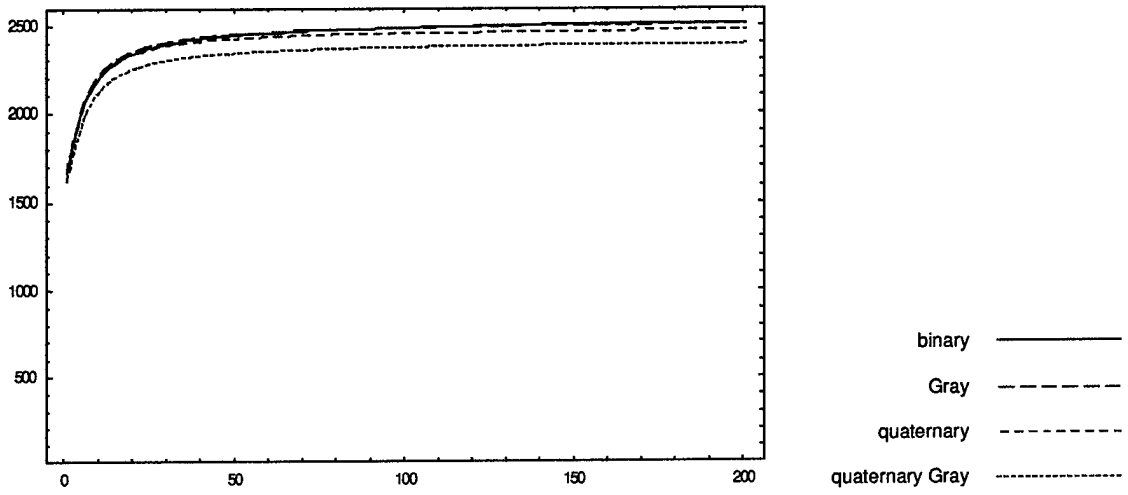


Figure 2: Offline-Performance of $EF1_3$

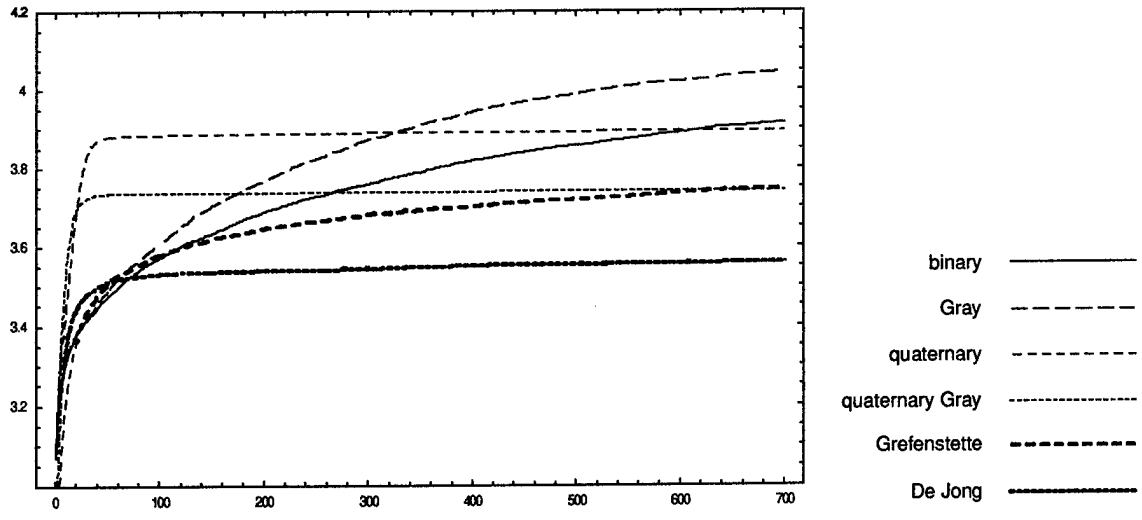


Figure 3: Offline-Performance of $EF3_5$

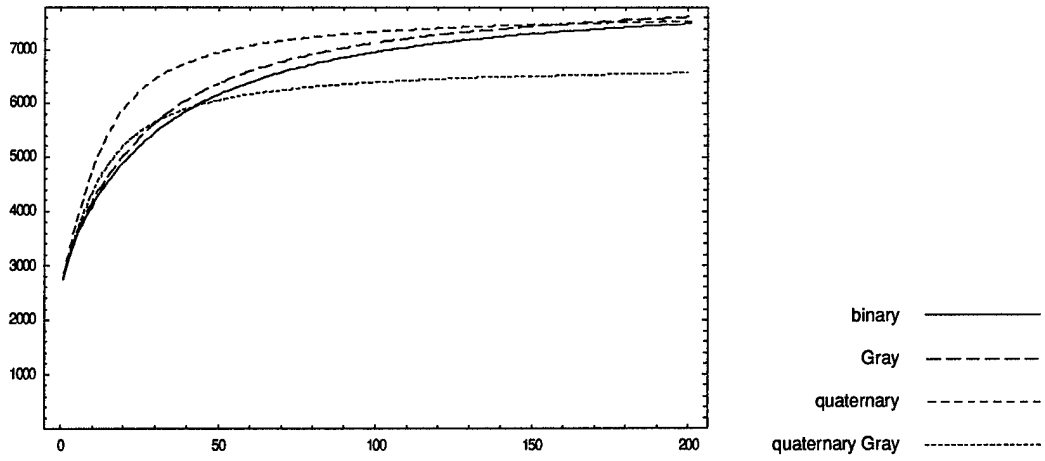


Figure 4: Highest fitness of $EF1_{10}$

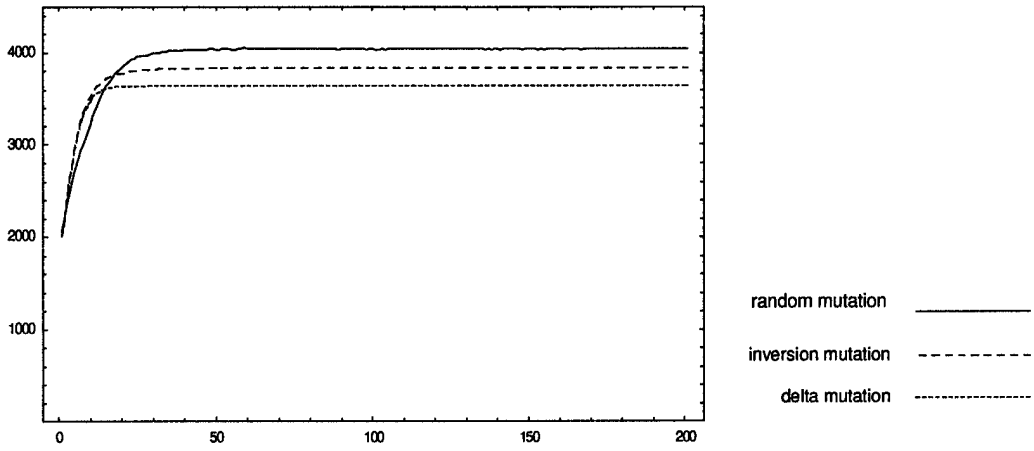


Figure 5: Influence of different mutation operators for highest fitness of $EF1_5$

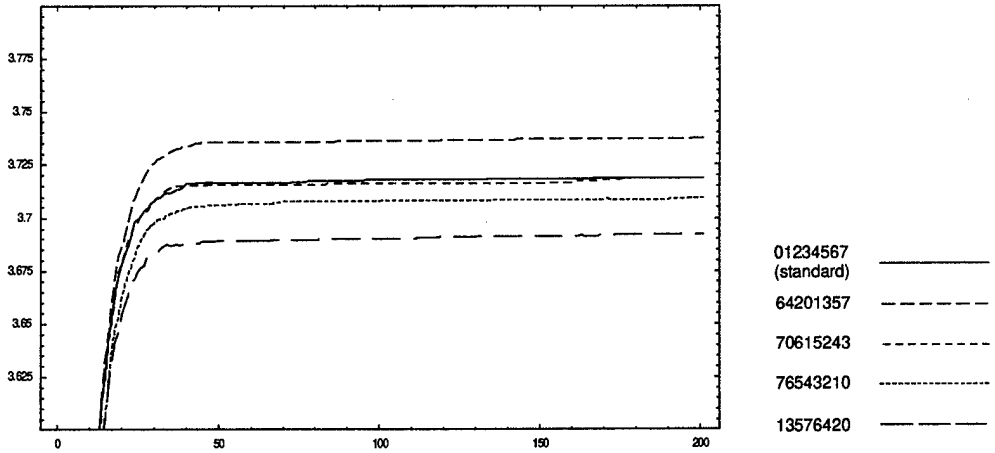


Figure 6: Influence of different Gray codes for highest fitness $EF3_5$

Redundant Complex Arithmetic and Its Application to Complex Multiplier Design

Takafumi Aoki, Ken-ichi Hoshi, and Tatsuo Higuchi
Department of System Information Sciences
Graduate School of Information Sciences, Tohoku University
Aoba-yama 05, Sendai 980-8579, Japan
aoki@ecei.tohoku.ac.jp

Abstract

This paper presents a class of complex number representations called Redundant Complex Number Systems (RCNSs), which are useful for designing VLSI signal processors with complex arithmetic capability. A redundant complex number system is defined as an imaginary-radix number system having a redundant integer digit set. This makes possible the construction of high-speed complex arithmetic circuits: examples include a complex-number parallel adder with no carry propagation chain, and a complex-number multiplier using fast binary-tree addition structure. This paper also presents the experimental fabrication of the RCNS-based complex multiplier in 0.5μm CMOS technology.

1. Introduction

Complex arithmetic computations are of major importance for various signal processing and scientific computation algorithms. Such algorithms include complex orthogonal transformations, convolutions, correlations and complex filtering. These applications require efficient representation and manipulation of complex numbers together with real numbers. To simplify the manipulation of complex numbers, several authors have proposed special number representations for complex numbers [1]–[4]. However, most of the studies focused on the theoretical aspects of complex number systems, and did not fully discuss the practical issues of VLSI circuit design and application.

Our goal is to develop a new complex number representation that makes possible the construction of high-speed complex arithmetic circuits which can be installed into practical VLSI signal processors. For this purpose, we extend the concept of "imaginary radix notation" in Knuth's *quarter-imaginary number system* [5], and propose a class of complex number representations called *Redundant Complex Number Systems* (RCNSs) (see [6]–[8] for earlier discussions on this topic[1]). An RCNS is a radix-rj system with digits in $\{-\alpha, \cdots, 0, \cdots, \alpha\}$, where $r(\geq 2)$ is an integer, j denotes the imaginary unit and $\lceil r^2/2 \rceil \leq \alpha \leq r^2 - 1$. The imaginary radix rj allows unified complex number representation without treating real and imaginary parts separately. Also, redundancy in number representation allows the carry-propagation-free addition and the binary-tree multiple-operand addition, as in Avizienis' Signed-Digit (SD) number systems [10].

This paper describes efficient designs of complex arithmetic circuits based on the radix-$2j$ RCNSs: examples include a complex-number parallel adder without carry propagation, and a complex-number multiplier using fast binary-tree multiple-operand addition structure together with an operand recoding scheme for reducing complex-number partial products. This paper also describes the result of implementing the proposed complex multiplier architecture in 0.5μm CMOS technology.

2. Redundant Complex Number Systems

2.1. Definition

A redundant complex number system (RCNS) is defined as a positional number system that has an imaginary radix rj, where r is an integer that is not less than 2 and j denotes the imaginary unit (i.e., $j^2 = -1$). Each digit of a redundant complex number can assume

[1]We presented the idea of redundant complex arithmetic in 1995. Nielsen and Muller also suggested similar possibility independently of our works. See [9] for their complex adder designs.

0-7695-0161-3/99 $10.00 © 1999 IEEE

the following $2\alpha + 1$ values

$$D_\alpha = \{\overline{\alpha}, \cdots, \overline{1}, 0, 1, \cdots, \alpha\}, \tag{1}$$

where $\overline{\alpha} = -\alpha$, and the maximum digit magnitude α must be within the following range:

$$\left\lceil \frac{r^2}{2} \right\rceil \leq \alpha \leq r^2 - 1. \tag{2}$$

The notation $\lceil x \rceil$ refers to the least integer that is not less than the real number x. An RCNS with radix rj and digit set D_α is denoted simply by RCNS rj, α. In the case of radix $2j$, for example, we can consider the two distinct systems: RCNS $2j, 3$ (with the digit set $D_3 = \{\overline{3}, \overline{2}, \overline{1}, 0, 1, 2, 3\}$) and RCNS $2j, 2$ (with the digit set $D_2 = \{\overline{2}, \overline{1}, 0, 1, 2\}$). The algebraic value of a number $\boldsymbol{X} = (X_{k-1} \cdots X_0 . X_{-1} \cdots X_{-l})$ in RCNS rj, α notation can be evaluated as

$$\boldsymbol{X} = \sum_{i=-l}^{k-1} X_i (rj)^i. \tag{3}$$

Example 1 Given an explicit value $1.25 - 1.5j$, let us consider the RCNS $2j, 3$ representation of the value with word length $k = 4$ and $l = 2$. There are twelve legitimate RCNS representations as follows:

$\overline{8j}$	$\overline{4}$	$2j$	1	$\overline{j/2}$	$\overline{1/4}$	$\overline{8j}$	$\overline{4}$	$2j$	1	$\overline{j/2}$	$\overline{1/4}$
0	0	$\overline{1}$	2 .	$\overline{1}$	3	0	0	0	2 .	3	3
1	0	3	2 .	$\overline{1}$	3	0	0	$\overline{1}$	1 .	$\overline{1}$	$\overline{1}$
0	0	0	1 .	3	$\overline{1}$	1	0	3	1 .	$\overline{1}$	$\overline{1}$
0	$\overline{1}$	$\overline{1}$	$\overline{2}$.	$\overline{1}$	3	0	$\overline{1}$	0	$\overline{2}$.	3	3
1	$\overline{1}$	3	$\overline{2}$.	$\overline{1}$	3	0	$\overline{1}$	$\overline{1}$	$\overline{3}$.	$\overline{1}$	$\overline{1}$
0	$\overline{1}$	0	$\overline{3}$.	3	$\overline{1}$	1	$\overline{1}$	3	$\overline{3}$.	$\overline{1}$	$\overline{1}$

□

There exists a close relationship between RCNSs and Signed-Digit (SD) number systems. That is, if we decompose the radix-rj redundant complex number representation into the real vector $(X_{2\lfloor (k-1)/2 \rfloor} \cdots X_0 \cdots X_{-2\lfloor l/2 \rfloor})$ and the imaginary vector $(X_{2\lceil (k-1)/2 \rceil - 1} \cdots X_1 X_{-1} \cdots X_{-2\lceil l/2 \rceil + 1})$, each representation can be regarded as a special Signed-Digit (SD) number representation that has the negative radix $-r^2$. Note here that the notation $\lfloor x \rfloor$ refers to the largest integer that is not greater than the real number x. Figure 1 summarizes distinctive features of RCNSs.

In this paper, we focus on the radix-$2j$ RCNSs, i.e., RCNS $2j, 3$ and RCNS $2j, 2$. The basic arithmetic circuits for radix-$2j$ RCNSs can be efficiently implemented with the state-of-the-art redundant binary architecture [11],[12] as discussed in Section 3. Another interesting choice of circuit technology is current-mode CMOS for radix-4 arithmetic [13].

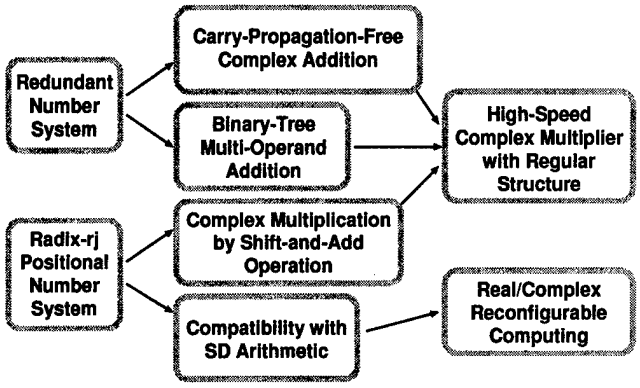


Figure 1. RCNS features for VLSI signal processing.

2.2. Addition in Radix-$2j$ RCNSs

The addition of two complex numbers, $\boldsymbol{X} = (X_{k-1} \cdots X_i \cdots X_{-l})$ and $\boldsymbol{Y} = (Y_{k-1} \cdots Y_i \cdots Y_{-l})$ in RCNS $2j, 3$, where $X_i, Y_i \in \{\overline{3}, \overline{2}, \overline{1}, 0, 1, 2, 3\}$, is performed by the following three successive steps for each digit:

$$\text{Step 1} \quad Z_i = X_i + Y_i, \tag{4}$$

$$\text{Step 2} \quad -4C_i + U_i = Z_i, \tag{5}$$

$$\text{Step 3} \quad S_i = U_i + C_{i-2}, \tag{6}$$

where Z_i is the *linear sum* (of X_i and Y_i), U_i is called the *intermediate sum*, and C_i is the *carry*. The ranges of the variables are

$$X_i, Y_i \in \{\overline{3}, \overline{2}, \overline{1}, 0, 1, 2, 3\}, \tag{7}$$

$$Z_i \in \{\overline{6}, \cdots, 0, \cdots, 6\}, \tag{8}$$

$$U_i \in \{\overline{2}, \overline{1}, 0, 1, 2\}, \tag{9}$$

$$C_i \in \{\overline{1}, 0, 1\}, \tag{10}$$

$$S_i \in \{\overline{3}, \overline{2}, \overline{1}, 0, 1, 2, 3\}. \tag{11}$$

The digit selection rule for Step 2 can be written as

$$\begin{aligned} U_i &= Z_i - 4, \quad C_i = \overline{1} \quad (\text{if } Z_i \geq 2), \\ U_i &= Z_i, \quad\quad C_i = 0 \quad (\text{if } \overline{2} \leq Z_i \leq 2), \\ U_i &= Z_i + 4, \quad C_i = 1 \quad (\text{if } Z_i \leq \overline{2}). \end{aligned}$$

Note that there are two possible choices for the values of U_i and C_i, when $Z_i = \pm 2$. The actual choice depends on the circuit implementation. In any case, the carry output C_i is determined from Z_i in Step 2 independently of the other carry signals. Thus totally parallel addition can be achieved with limited carry propagation. Figure 2 shows an example of RCNS $2j, 3$ addition. This type of carry-propagation-free addition is also possible in RCNS $2j, 2$.

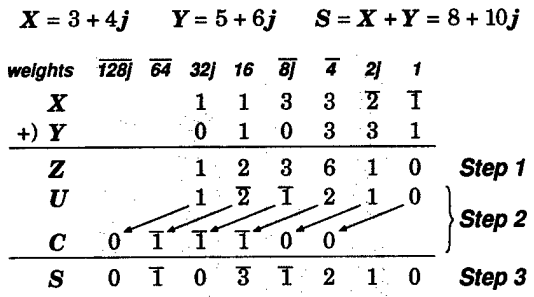


Figure 2. RCNS $2j$,3 addition.

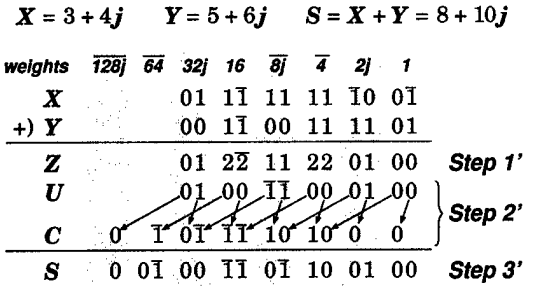


Figure 3. RCNS $2j$,3 addition using binary SD coding.

Table 1. RCNS $2j$,3 digit representation.

RCNS X_i	$\overline{3}$	$\overline{2}$	$\overline{1}$	0	1	2	3
Binary SD $x_i^1\ x_i^0$	$\overline{1}\,\overline{1}$	$\overline{1}\,0$	$0\,\overline{1}$ $\overline{1}\,1$	$0\,0$	$0\,1$ $1\,\overline{1}$	$1\,0$	$1\,1$

3. Arithmetic Circuits Design

3.1. RCNS $2j$,3 Adder Using Binary SD Coding

As it will become apparent later on, RCNS $2j,3$ is useful for fast complex-number addition because of its close relationship with the binary SD number system[2]. This section describes the design of a complex-number parallel adder circuit using binary SD coding of RCNS $2j,3$ digit. On the other hand, RCNS $2j,2$ is particularly useful for multiplier recoding algorithm as described in Section 4.

Let $\boldsymbol{X} = (X_{k-1} \cdots X_i \cdots X_{-l})$ be a complex number in RCNS $2j,3$ notation. When RCNS $2j,3$ arithmetic circuits are constructed with binary logic elements, each digit X_i ($\in \{\overline{3}, \overline{2}, \overline{1}, 0, 1, 2, 3\}$) must be coded into a vector of binary bits. We introduce here binary SD coding of the RCNS $2j,3$ digit. In this coding scheme, the RCNS $2j,3$ digit X_i is represented by a 2-digit binary SD code $x_i^1 x_i^0$ as follows:

$$X_i = 2x_i^1 + x_i^0 \quad (x_i^1, x_i^0 \in \{\overline{1}, 0, 1\}). \quad (12)$$

Table 1 illustrates this coding. Using the above coding, RCNS $2j,3$ addition process (4)–(6) can be decomposed as

$$\text{Step } 1': \quad z_i^1 = x_i^1 + y_i^1, \qquad z_i^0 = x_i^0 + y_i^0, \quad (13)$$

$$\text{Step } 2': \quad -2c_i^1 + u_i^1 = z_i^1, \quad 2c_i^0 + u_i^0 = z_i^0, \quad (14)$$

$$\text{Step } 3': \quad s_i^1 = u_i^1 + c_i^0, \qquad s_i^0 = u_i^0 + c_{i-2}^1. \quad (15)$$

[2]In this paper, we use the term "binary SD number system" instead of "redundant binary number system" in order to avoid misinterpretation.

The ranges of the variables are given by $x_i^\delta, y_i^\delta \in \{\overline{1}, 0, 1\}$, $z_i^\delta \in \{\overline{2}, \overline{1}, 0, 1, 2\}$, $u_i^\delta \in \{\overline{1}, 0, 1\}$, $c_i^\delta \in \{\overline{1}, 0, 1\}$, and $s_i^\delta \in \{\overline{1}, 0, 1\}$, where $\delta \in \{0, 1\}$. In Step 2', the values of u_i^δ and c_i^δ are selected in order that the final sum s_i^δ is kept within the range of $\{\overline{1}, 0, 1\}$. This can be done by using the well-known addition algorithm of binary SD numbers [11],[14] with a slight modification at the sign of upper carry digit c_i^1. The values of u_i^δ and c_i^δ are selected according to the sign of the linear sum z_i^0 (for $\delta = 1$) or z_{i-2}^1 (for $\delta = 0$). Figure 3 shows an example of the binary-SD-coded RCNS $2j,3$ addition, which is equivalent to the example of Figure 2. The calculation for a single digit in RCNS $2j,3$ addition can thus be performed by using a pair of binary SD full adders (or redundant binary full adders). Figure 4 illustrates the RCNS parallel adder realization using binary SD full adders. Thus, the well-known binary SD architecture [11],[12] can be employed for implementing RCNS $2j,3$ arithmetic.

Figure 5 shows the proposed configuration of RCNS $2j,3$ full adder circuit using a pair of binary SD full adders. We have derived a new circuit realization of a high-speed binary SD full adder cell by transforming the 4-2 compressor [15] using dual-rail pass transistor logic. Using the proposed RCNS $2j,3$ full adder circuit, a high-speed complex-number parallel adder with no carry propagation chain has been derived.

3.2. Vector Notation for Algorithm Description

This section introduces useful notation for defining number representation systems, which is an extension of Koren's notation [14]. We use the triplet $\langle D, \boldsymbol{\Lambda}, \boldsymbol{W} \rangle$ to identify a positional number system of $k+l$ digits (k digits in the integral part and l digits in the fractional parts), where D is the *digit set*, $\boldsymbol{\Lambda} = (\lambda_{k-1}, \cdots, \lambda_i, \cdots, \lambda_{-l})$ is the *sign vector*, and $\boldsymbol{W} = (w_{k-1}, \cdots, w_i, \cdots, w_{-l})$ is the *absolute weight vector*, respectively. The algebraic value X of a $(k+l)$-digit vector $(X_{k-1} \cdots X_0 . X_{-1} \cdots X_{-l})$ in the system

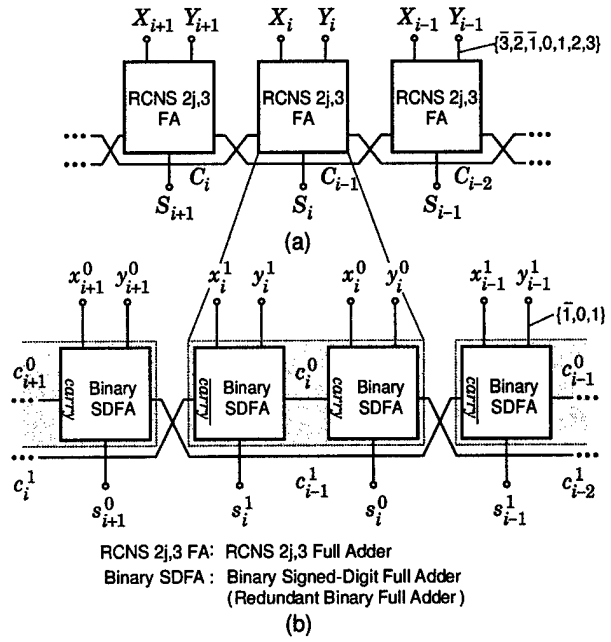


Figure 4. Realization of an RCNS parallel adder using binary SDFAs.

$\langle D, \boldsymbol{\Lambda}, \boldsymbol{W} \rangle$ is given by

$$X = \sum_{i=-l}^{k-1} X_i \lambda_i w_i, \tag{16}$$

where w_i is the absolute value of the weight at the ith position, and $\lambda_i \in \{1, \bar{1}, j, \bar{j}\}$.

Using the triple notation, we can represent RCNS $2j,3$ of 4 digits ($k = 3$, $l = 1$) by $\langle D_{RCNS}, \boldsymbol{\Lambda}_{RCNS}, \boldsymbol{W}_{RCNS} \rangle$, where

$$\begin{array}{lcl} D_{RCNS} & = & \{ \bar{3},\ \bar{2},\ \bar{1},\ 0,\ 1, 2, 3 \}, \\ \Lambda_{RCNS} & = & (\ \bar{1},\ j,\ 1,\ \bar{j}\ ), \\ W_{RCNS} & = & (\ 2^2, 2^1, 2^0, 2^{-1}\ ). \end{array} \tag{17}$$

As mentioned in the last section, the circuit implementation of RCNS $2j,3$ is based on the decomposition of RCNS $2j,3$ into binary SD number representation. By decomposing every digit of the above triple notation into 2-digit binary SD code, we have the system $\langle D_I, \boldsymbol{\Lambda}_I, \boldsymbol{W}_I \rangle$, where

$$\begin{array}{lcl} D_I & = & \{\ \bar{1},\ 0,\ 1\ \}, \\ \Lambda_I & = & (\ \bar{1},\ \bar{1},\ j,\ j,\ 1,\ 1,\ \bar{j},\ \bar{j}\ ), \\ W_I & = & (\ 2^3, 2^2, 2^2, 2^1, 2^1, 2^0, 2^0, 2^{-1}\ ). \end{array} \tag{18}$$

The complex-number parallel adder of Figure 4 (b) executes the operation in this number system.

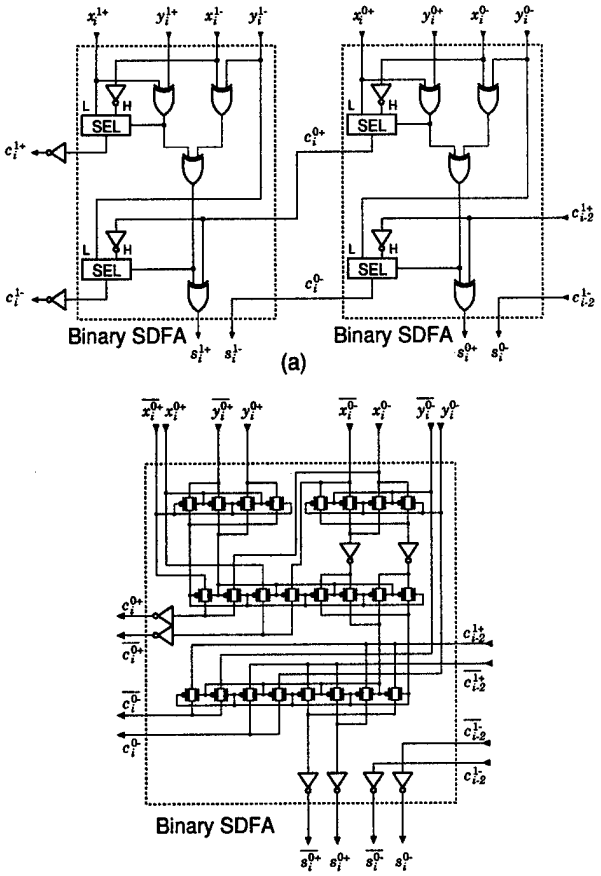


Figure 5. RCNS $2j,3$ full adder: (a) logical circuit diagram, (b) circuit details of the binary SDFA cell.

4. High-Speed Complex Multiplier

4.1. Complex Multiplication Algorithm

This section is to design a complex-number multiplier, that can be installed into practical VLSI signal processors. The input/output numbers of this multiplier are assumed to be in the merged two's complement binary form $\langle D_{m2c}, \boldsymbol{\Lambda}_{m2c}, \boldsymbol{W}_{m2c} \rangle$. (To obtain this representation, we first represent the real part and the imaginary part of the input operand by two's complement number system, and merge the real and imaginary strings into one string.) We also assume that both the multiplier $\boldsymbol{X}$ and the multiplicand $\boldsymbol{Y}$ are given with the precision of (n,n) bits[3]. Figure 6 shows an example of the total process of $(4,4) \times (4,4)$-bit complex-number multiplication.

The (n,n)-bit multiplicand $\boldsymbol{X}$ is first given in the

[3]The symbol (n_1, n_2) represents the precision of complex numbers, where n_1 is for real part and n_2 is for imaginary part.

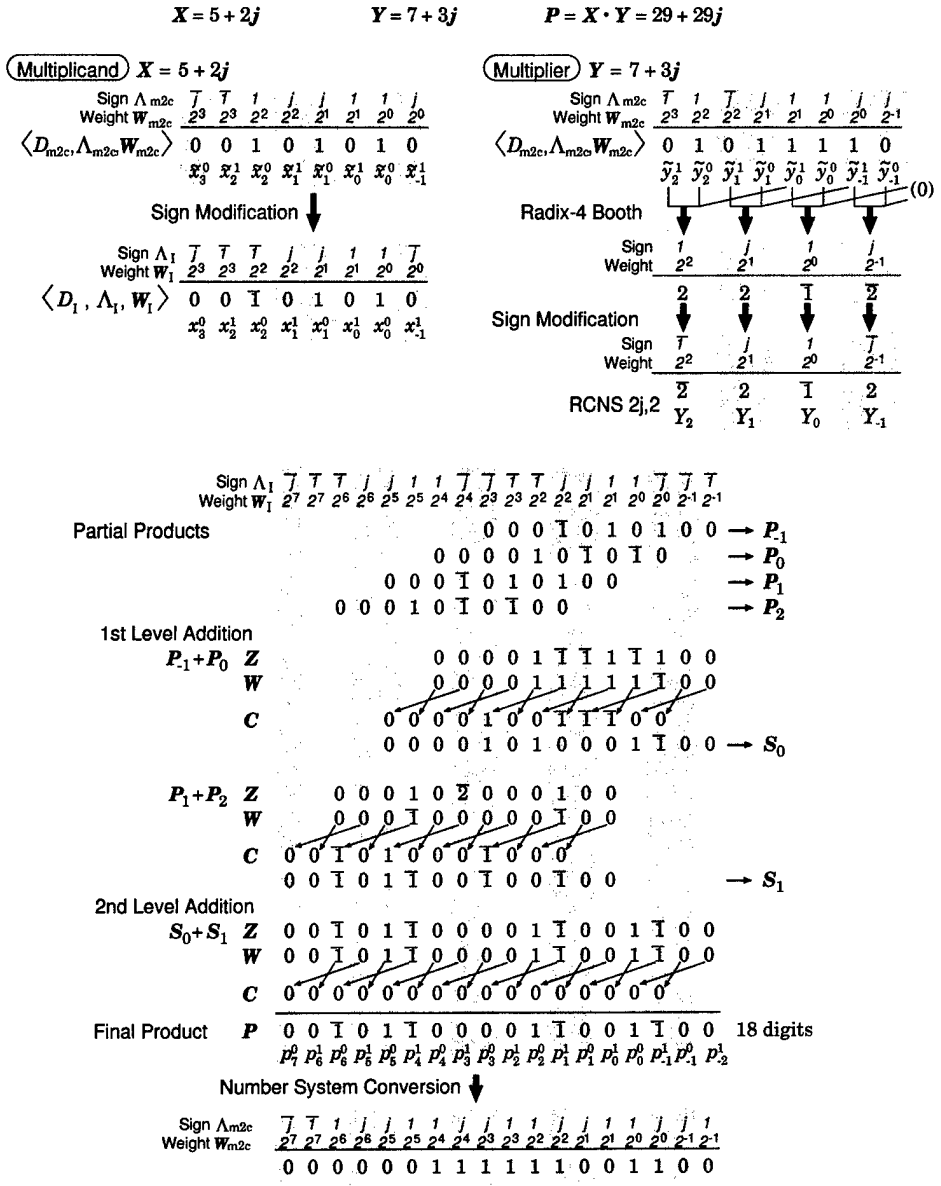


Figure 6. An example of $(4,4) \times (4,4)$-bit complex-number multiplication.

form of $\langle\, D_{m2c}, \boldsymbol{\Lambda}_{m2c}, \boldsymbol{W}_{m2c} \,\rangle$ as

Λ_{m2c}	$\bar{j}$	$\bar{1}$	1	j	$\cdots$	1	1	j
W_{m2c}	2^{n-1}	2^{n-1}	2^{n-2}	2^{n-2}	$\cdots$	2^1	2^0	2^0
	$\tilde{x}^0_{n-1}$	$\tilde{x}^1_{n-2}$	$\tilde{x}^0_{n-2}$	$\tilde{x}^1_{n-3}$	$\cdots$	$\tilde{x}^1_0$	$\tilde{x}^0_0$	$\tilde{x}^1_{-1}$,

(19)

where each digit is in $D_{m2c}(= \{0,1\})$ and we assume n is even without loss of generality. This multiplicand is converted into $\langle\, D_I, \boldsymbol{\Lambda}_I, \boldsymbol{W}_I \,\rangle$ expression:

Λ_I	j^{n-1}	j^{n-1}	$\cdots$	j^s	j^s	$\cdots$	j^0	j^0	j^{-1}	j^{-1}
W_I	2^n	2^{n-1}	$\cdots$	2^{s+1}	2^s	$\cdots$	2^1	2^0	2^0	2^{-1}
		x^0_{n-1}	$\cdots$	x^1_s	x^0_s	$\cdots$	x^1_0	x^0_0	x^1_{-1}	.

(20)

On the other hand, the (n,n)-bit multiplier $\boldsymbol{Y}$ in $\langle\, D_{m2c}, \boldsymbol{\Lambda}_{m2c}, \boldsymbol{W}_{m2c} \,\rangle$ is converted into the RCNS $2j, 2$ representation in order to reduce the number of partial products. This multiplier recoding can be done by using the well-known radix-4 modified Booth's algorithm [14] as follows: Let the (n,n)-bit multiplier $\boldsymbol{Y}$ in $\langle\, D_{m2c}, \boldsymbol{\Lambda}_{m2c}, \boldsymbol{W}_{m2c} \,\rangle$ be given by

Λ_{m2c}	$\bar{1}$	1	$\bar{j}$	j	$\cdots$	1	1	j	j
W_{m2c}	2^{n-1}	2^{n-2}	2^{n-2}	2^{n-3}	$\cdots$	2^1	2^0	2^0	2^{-1}
	$\tilde{y}^1_{n-2}$	$\tilde{y}^0_{n-2}$	$\tilde{y}^1_{n-3}$	$\tilde{y}^0_{n-3}$	$\cdots$	$\tilde{y}^1_0$	$\tilde{y}^0_0$	$\tilde{y}^1_{-1}$	$\tilde{y}^0_{-1}$,

(21)

where we assume that the range of the n-bit imaginary part is shifted one bit relative to that of the real part in order to simplify the algorithm description. Thus, the least significant imaginary digit has the weight of 2^{-1}. (One can easily adjust the ranges by introduc-

ing a dummy digit at the most significant position.) Recoding the multiplier with the equation:

$$Y'_t = -2\tilde{y}^1_t + \tilde{y}^0_t + \tilde{y}^1_{t-2}, \tag{22}$$

where $t = -1, 0, 1, 2, \cdots, n-2$ and $\tilde{y}^1_{-2} = \tilde{y}^1_{-3} = 0$, we can rewrite the multiplier in the form:

Sign	1	j	$\cdots$	j	1	j
Weight	2^{n-2}	2^{n-3}	$\cdots$	2^1	2^0	2^{-1}
	Y'_{n-2}	Y'_{n-3}	$\cdots$	Y'_1	Y'_0	Y'_{-1}

, (23)

where $Y'_t \in \{\overline{2}, \overline{1}, 0, 1, 2\}$. Modifying the sign vector,

Sign	j^{n-2}	j^{n-3}	$\cdots$	j^1	j^0	j^{-1}
Weight	2^{n-2}	2^{n-3}	$\cdots$	2^1	2^0	2^{-1}
	Y_{n-2}	Y_{n-3}	$\cdots$	Y_1	Y_0	Y_{-1}

, (24)

where $Y_t \in \{\overline{2}, \overline{1}, 0, 1, 2\}$. This is the fixed radix number system corresponding to RCNS $2j, 2$. Thus, the multiplier $\boldsymbol{Y}$ is recoded into the RCNS $2j, 2$ vector $(Y_{n-2} \cdots Y_t \cdots Y_{-1})$ given by

$$\boldsymbol{Y} = \sum_{t=-1}^{n-2} Y_t \times (2j)^t. \tag{25}$$

The multiplicand $\boldsymbol{X}$ given by (20) and the recoded multiplier $\boldsymbol{Y}$ given by (25) generate n partial products $\boldsymbol{P}_{n-2}, \cdots, \boldsymbol{P}_{-1}$ as

$$\boldsymbol{P}_t = \boldsymbol{X} \cdot Y_t \times (2j)^t. \tag{26}$$

Note that $\boldsymbol{X} \times (2j)^t$ corresponds to the $2t$-digit shift of the vector $\boldsymbol{X}$ in the $\langle D_I, \boldsymbol{\Lambda}_I, \boldsymbol{W}_I \rangle$ notation, since this system is "decomposed" version of RCNS $2j, 3$. Hence, $\boldsymbol{P}_t$ can be represented in $\langle D_I, \boldsymbol{\Lambda}_I, \boldsymbol{W}_I \rangle$ as

Λ_I	j^{n+t}	j^{n+t}	$\cdots$	j^{s+t}	j^{s+t}	$\cdots$	j^t	j^t	j^{-1+t}	j^{-1+t}
W_I	2^{n+1+t}	2^{n+t}	$\cdots$	2^{s+1+t}	2^{s+t}	$\cdots$	2^{1+t}	2^t	2^t	2^{-1+t}
		p^0_{nt}	$\cdots$	p^1_{st}	p^0_{st}	$\cdots$	p^1_{0t}	p^0_{0t}	p^1_{-1t}	

. (27)

where $p^\delta_{st} \in \{\overline{1}, 0, 1\}$ and $t = -1, \cdots, n-2$. Each digit of the partial products is calculated by shift and complement operations according to the value of Y_t as

$$\begin{aligned}
&\text{if } Y_t = 2, \quad \text{then } p^1_{st} = x^0_s,\ p^0_{st} = \overline{x^1_{s-2}},\\
&\text{if } Y_t = 1, \quad \text{then } p^1_{st} = x^1_s,\ p^0_{st} = x^0_s,\\
&\text{if } Y_t = 0, \quad \text{then } p^1_{st} = p^0_{st} = 0,\\
&\text{if } Y_t = \overline{1}, \quad \text{then } p^1_{st} = \overline{x^1_s},\ p^0_{st} = \overline{x^0_s},\\
&\text{if } Y_t = \overline{2}, \quad \text{then } p^1_{st} = \overline{x^0_s},\ p^0_{st} = x^1_{s-2}.
\end{aligned}$$

These partial products $\boldsymbol{P}_{n-2}, \cdots, \boldsymbol{P}_{-1}$ are in the form of $\langle D_I, \boldsymbol{\Lambda}_I, \boldsymbol{W}_I \rangle$. The next step is to accumulate all these partial products to obtain a single product $\boldsymbol{P}$ using RCNS parallel adder shown in Figure 4 (b). Since the RCNS parallel adder is regarded as a 2-to-1 reducer, a highly regular binary-tree structure of $\lceil \log_2 n \rceil$ levels can be employed for high-speed accumulation of the n partial products. The result of the adder tree is the $(4n+2)$-digit product $\boldsymbol{P}$ in $\langle D_I, \boldsymbol{\Lambda}_I, \boldsymbol{W}_I \rangle$ and consequently a conversion to the merged two's complement representation $\langle D_{m2c}, \boldsymbol{\Lambda}_{m2c}, \boldsymbol{W}_{m2c} \rangle$ is needed, in most cases. This conversion can be performed by subtracting the digits having negative sign elements from the digits having positive sign elements for real/imaginary components independently. (This conversion process is similar to the well-known SD-to-binary conversion.) In the case of $(n, n) \times (n, n)$-bit multiplier, the conversion can be performed by employing a pair of $(2n+1)$-bit carry propagate adders for the real and imaginary parts independently.

4.2. Implementation and Evaluation

The block diagram of the proposed complex multiplier is illustrated in Figure 7 for $(4, 4) \times (4, 4)$-bit multiplication. Using the RCNS representation, we can construct a complex-number multiplier exhibiting a highly regular topological structure that resembles those of real-number multipliers. Table 2 compares real-number multipliers using the binary SD number representation and complex-number multipliers using the RCNS representation in terms of hardware complexity and the number of adder tree stages. In general, the $(n, n) \times (n, n)$-bit complex-number multiplier exhibits the same level of circuit complexity as that of the $2n \times 2n$-bit real-number multiplier.

In order to confirm the proposed principle, an $(8, 8) \times (8, 8)$-bit redundant complex multiplier has been designed and implemented in 0.5 μm standard CMOS technology. Figure 8 shows the chip micrograph and its main features. The chip integrates 15,340 transistors in an active area 5.98 mm^2. The computation time could be further improved to the same level with state-of-the-art real-number multipliers in principle, since there is no essential difference between the proposed multiplier configuration and the conventional real-number multipliers using binary SD (redundant binary) adders [12] or 4-2 compressors [15].

When a complex multiplier is implemented on the basis of conventional real-number arithmetic, four multipliers and two adders must be connected with complicated global interconnections. The use of RCNS makes possible the construction of a regular complex multiplier without using such global interconnections. We have compared the proposed design with the complex-number multiplier consisting of four real-number multipliers and two real-number adders using redundant bi-

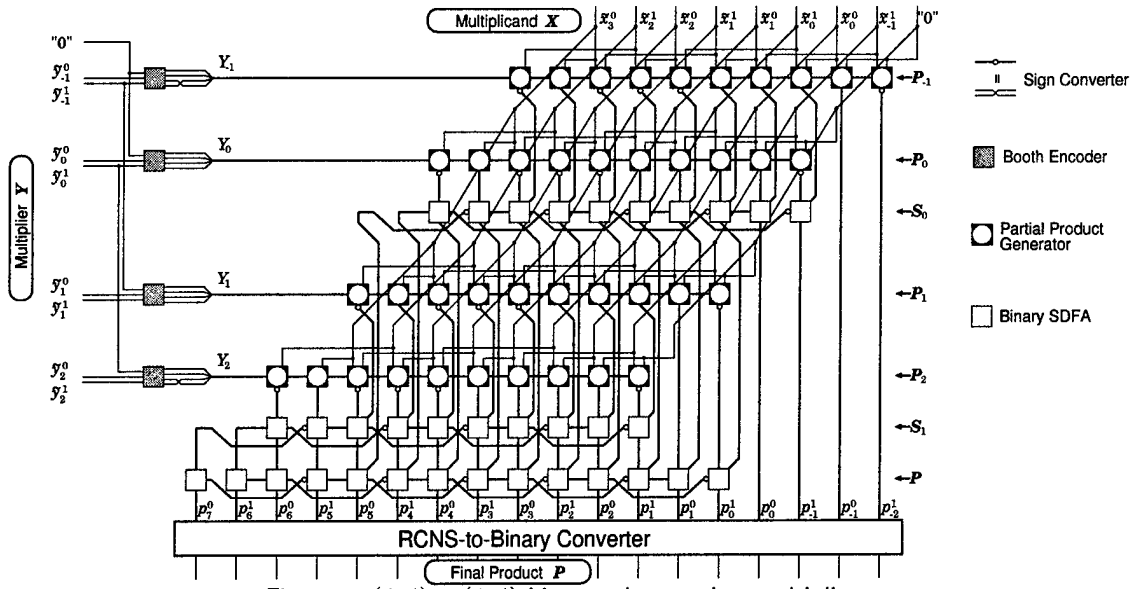
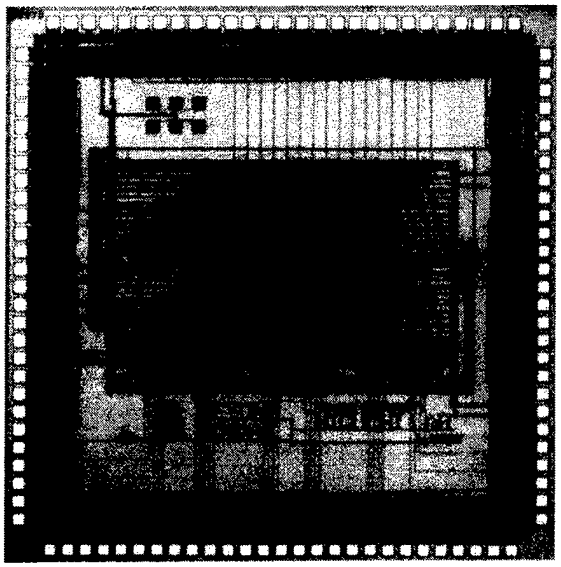


Figure 7. $(4,4) \times (4,4)$-bit complex-number multiplier.



Supply Voltage: **3.3 V**

Execution Time: **12.9 ns**

Active Area Size:
1.87 mm X 3.20 mm
(5.98 mm^2)

Transistor Count:
15,340 (except I/O)

Technology:
VDEC 0.5 um CMOS
Double Metal

Figure 8. $(8,8) \times (8,8)$-bit complex-number multiplier chip.

nary arithmetic. It is estimated that the RCNS-based design achieves 21% reduction in chip area for $(8,8)$-bit precision, and 27% reduction for $(32,32)$-bit precision. Also, the elimination of global interconnections makes possible significant reduction in wiring capacitance, which leads to high-speed low-power operation.

5. Conclusions

This paper presents the redundant complex arithmetic for VLSI signal processing, and its application to complex multiplier design. Another unique property of the redundant complex arithmetic is its compatibility with the Signed-Digit (SD) arithmetic, which allows us to design low-overhead real/complex reconfigurable arithmetic circuits [8]. As an example, we have also developed a real/complex reconfigurable arithmetic unit, which can change its structure for three different arithmetic modes in real time. The detailed result will be reported in future papers.

Acknowledgments The complex multiplier chip has been fabricated through the chip fabrication program of VDEC, the University of Tokyo with the collaboration by NTT Electronics Corporation and Dai Nippon Printing Corporation.

Table 2. Comparison of real-number multipliers and complex-number multipliers.

	Operand Wordlength (bit)	Number of Modules*				Number of Transistors**	Number of Inter-Module Interconnects	Number of Adder Tree Levels
		BE	PPG	SDFA	CPA			
Real-Number Multipliers	8 × 8	4	36	30	16 bit × 1	3862	428	2
	16 × 16	8	136	132	32 bit × 1	14198	1688	3
	32 × 32	16	528	537	64 bit × 1	53382	6588	4
	64 × 64	32	2080	2133	128 bit × 1	203558	25752	5
	128 × 128	64	8256	8432	256 bit × 1	788774	101228	6
Complex-Number Multipliers	(4,4)×(4,4)	4	40	34	9 bit × 2	4328	474	2
	(8,8)×(8,8)	8	144	142	17 bit × 2	15340	1794	3
	(16,16)×(16,16)	16	544	560	33 bit × 2	55500	6826	4
	(32,32)×(32,32)	32	2112	2184	65 bit × 2	208300	26274	5
	(64,64)×(64,64)	64	8320	8542	129 bit × 2	798956	102346	6

*BE: Booth Encoder, PPG: Partial Product Generator, SDFA: Binary SD Full Adder, CPA: Carry Propagate Adder (for Number System Conversion)

**The designs are based on standard CMOS technology.

References

[1] I. Koren and Y. Maliniak,"On classes of positive, negative, and imaginary radix number systems," *IEEE Trans. Computer*, Vol. C-30, No. 5, pp. 312–317, May 1981.

[2] T. T. Dao,"Knuth's complex arithmetic with quaternary hardware," *Proc. 12th IEEE Int'l Symp. Multiple-Valued Logic*, pp. 94–98, May 1982.

[3] J. Duprat, Y. Herreros, and S. Kla,"New redundant representations of complex numbers and vectors," *IEEE Trans. Computers*, Vol. C-42, No. 7, pp. 817–824, July 1993.

[4] A. M. Nielsen and P. Kornerup,"On radix representation of rings," *Proc. 13th IEEE Int'l Symp. Computer Arithmetic*, pp. 34–43, July 1997.

[5] D. E. Knuth,*The Art of Computer Programming*, Vol. 2,Reading, MA: Addison-Wesley Publishing Co., 1973.

[6] Y. Ohi, T. Aoki, and T. Higuchi,"Redundant complex number systems," *Proc. 25th IEEE Int'l Symp. Multiple-Valued Logic*, pp. 14–19, May 1995.

[7] T. Aoki, Y. Ohi, and T. Higuchi,"Redundant complex number arithmetic for high-speed signal processing," *VLSI SIGNAL PROCESSING VIII (1995 IEEE Workshop on VLSI Signal Processing)*, pp. 523–532, October 1995.

[8] T. Aoki, H. Amada, and T. Higuchi, "Real/complex reconfigurable arithmetic using redundant complex number systems," *Proc. of the 13th IEEE Symposium on Computer Arithmetic*, pp. 200–207, July 1997.

[9] A. M. Nielsen and J-M. Muller,"Borrow-save adders for real and complex number systems," *Proc. 2nd Conf. on Real Numbers and Computers*, April 1996.

[10] A. Avizienis,"Signed-digit number representations for fast parallel arithmetic," *IRE Trans. Electronic Computers*, Vol. EC-10, pp. 389–400, September 1961.

[11] N. Takagi, H. Yasuura, and S. Yajima,"High-speed VLSI multiplication algorithm with a redundant binary addition tree," *IEEE Trans. Computer*, Vol. C-34, No. 9, pp. 789–796, September 1985.

[12] H. Makino, Y. Nakase, H. Suzuki, H. Morinaka, H. Shinohara, and K. Mashiko,"An 8.8-ns 54×54-bit multiplier with high speed redundant binary architecture," *IEEE J. Solid-State Circuits*, Vol. 31, No. 6, pp. 773–783, June 1996.

[13] S. Kawahito, M. Kameyama, T. Higuchi, and H. Yamada,"A 32×32-bit multiplier using multiple-valued MOS current-mode circuits," *IEEE J. Solid-State Circuits*, Vol. SC-23, No. 1, pp. 124–132, February 1988.

[14] I. Koren, *Computer Arithmetic Algorithms*, Englewood Cliffs, NJ: Prentice-Hall, 1993.

[15] N. Ohkubo, M. Suzuki, T. Shinbo, T. Yamanaka, A. Shimizu, K. Sasaki, and Y. Nakagome,"A 4.4ns CMOS 54×54-b multiplier using pass-transistor multiplexer," *IEEE J. Solid-State Circuits*, Vol. 30, No. 3, pp. 251–257, March 1995.

On the Number of Multilinear Partitions and the Computing Capacity of Multiple-Valued Multiple-Threshold Perceptrons

Alioune Ngom * Ivan Stojmenović † Joviša Žunić ‡

Abstract

We introduce the concept of multilinear partition of a point set $V \subset R^n$ and the concept of multilinear separability of a function $f : V \mapsto K = \{0, \ldots, k-1\}$. Based on well known relationships between linear partitions and minimal pairs, we derive formulae for the number of multilinear partitions of a point set in general position and of the set K^2. The (n, k, s)-perceptrons partition the input space V into $s+1$ regions with s parallel hyperplanes. We obtain results on the capacity of a single (n, k, s)-perceptron, respectively for $V \subset R^n$ in general position and for $V = K^2$. Finally, we describe a fast polynomial-time algorithm for counting the multilinear partitions of K^2.

1 Introduction

Let $V \subset R^n$ and $K = \{0, \ldots, k-1\}$, with $k \geq 2$. A *n-input k-valued real function* $f : V \mapsto K$ is a function with real-valued inputs and k-valued outputs. When $V \subseteq K^n$, we will refer to f as a *n-input k-valued logic function*. We denote by P_k^n the set of all n-input k-valued real (resp. logic) functions $f : V \mapsto K$ and by $P_k = \bigcup_{n \geq 1} P_k^n$ the set of all k-valued real (logic) functions. For instance, for $k = 2$ P_2 is the set of all binary logic functions.

A n-input k-valued s-threshold perceptron [3], abbreviated as *(n, k, s)-perceptron*, computes a *n-input k-valued weighted s-threshold function* $F_{k,s}^n(\vec{w}, \vec{t}, \vec{o})$ given by

$$F_{k,s}^n(\vec{w}, \vec{t}, \vec{o})(\vec{x}) = \begin{cases} o_0 & \text{if } \vec{w}\vec{x} < t_1 \\ o_i & \text{if } t_i \leq \vec{w}\vec{x} < t_{i+1} \ 1 \leq i \leq s-1 \\ o_s & \text{if } t_s \leq \vec{w}\vec{x} \end{cases}$$

where input vector $\vec{x} = (x_1, \ldots, x_n) \in V$, weight vector $\vec{w} = (w_1, \ldots, w_n) \in V$, threshold vector $\vec{t} = (t_1, \ldots, t_s) \in R^s$ and $t_i \leq t_{i+1}$ $(1 \leq i \leq s-1$ and $1 \leq s \leq k^n - 1$, the number of threshold values), and output vector $\vec{o} = (o_0, \ldots, o_s) \in K^{s+1}$. A (n, k, s)-perceptron simulates a k-valued function $f : V \mapsto K$. Depending on V we will either refer to real or logic (n, k, s)-perceptrons. It is well known that any n-input k-valued logic function can be transformed into a k-valued s-threshold function [1] for some s.

Multiple-threshold devices [1] are threshold elements containing multiple levels of excitation (thresholds). They have drawn less enthousiasm. Among there qualities, though, is that given enough thresholds, a single multiple-threshold element can realize any given function operating on a finite domain.

The $(n, k, k-1)$-perceptrons were introduced by [4] as model for simulating continuous perceptrons with limited precision on their inputs. Learning algorithms for some classes of (n, k, s)-perceptrons are investigated in [3]. [7] introduced the $(n, 2, s)$-perceptrons and their computational power have been intensively studied in [5].

In this research paper we propose to compute the *capacity* of a (n, k, s)-perceptron, that is, the number of k-valued functions $f : V \mapsto K$ that can be simulated by a single (n, k, s)-perceptron.

There has been an intensive interest in threshold logic as the main component of neural network models. These models provide a direction for pattern recognition systems with distinct natural advantages. The capacity of these models, as well as their computing power, are directly related to the number of threshold functions. The ability of multiple-threshold devices to simulate a larger number of functions compared to single-threshold devices is vital for the capacity and capabilities of neural network models based on threshold

*Computer Science Department, Lakehead University, 855 Oliver Road, Thunder Bay, Ontario P7B 5E1, Canada, angom@ice.lakeheadu.ca. Research is partially supported by NSERC

†Computer Science Department, School of Information Technology and Engineering, University of Ottawa, 150 Louis Pasteur, Ottawa, Ontario K1N 9B4, Canada, ivan@csi.uottawa.ca. Research is partially supported by NSERC

‡Institute of Applied Basic Disciplines, Faculty of Engineering, University of Novi Sad, Veljka Vlahovića 3, 21000 Novi Sad, Yugoslavia

0-7695-0161-3/99 $10.00 © 1999 IEEE

logic. It is therefore of practical as well as theoretical interest to estimate the number of functions that can be modeled as multiple-threshold functions for a given number of inputs and threshold levels.

2 Multilinear separability and multilinear partition

The problem of computing (or simulating) a given function f by a (n,k,s)-perceptron, is to determine s and a vector $\vec{r} = (\vec{w},\vec{t},\vec{o}) \in R^{n+s} \times K^{s+1}$ such that $F^n_{k,s}(\vec{r})(\vec{x}) = f(\vec{x})$ $(\forall \vec{x} \in V)$, i.e. $f = F^n_{k,s}(\vec{r})$. We will refer to $\vec{r}$ as a *s-representation* of $F^n_{k,s}$ for f.

Let $V \subseteq R^n$ with cardinality $|V| = v \geq 1$. A k-valued function $f : V \mapsto K$ is *s-separable* if and only if it has a s-representation $(\vec{w},\vec{t},\vec{o})$. In other words, a (n,k,s)-perceptron partitions the space $V \subset R^n$ into $s+1$ distinct classes $H_0^{[o_0]},\ldots,H_s^{[o_s]}$, using s parallel hyperplanes, where $H_j^{[o_j]} = \{\vec{x} \in V | f(\vec{x}) = o_j$ and $t_j \leq \vec{w}\vec{x} < t_{j+1}\}$. We assume that $t_0 = -\infty$ and $t_{s+1} = +\infty$. Each hyperplane equation denoted by H_j $(1 \leq j \leq s)$ is of the form

$$H_j : \vec{w}\vec{x} = t_j$$

Multilinear separability (s-separability) extends the concept of *linear separability* (1-separability of the common binary 1-threshold perceptron) to the (n,k,s)-perceptron. Linear separability in two-valued case tells us that a $(n,2,1)$-perceptron can only learn from a space $V \subseteq [0,1]^n$ in which there is a single hyperplane which separates it into two disjoint halfspaces: $H_0^{[0]} = \{\vec{x}|f(\vec{x}) = 0\}$ and $H_1^{[1]} = \{\vec{x}|f(\vec{x}) = 1\}$. From the *$(n,2,1)$-perceptron convergence theorem* [6], concepts which are linearly nonseparable cannot be learned by a $(n,2,1)$-perceptron. One example of linearly nonseparable two-valued logic function is the n-input parity function. Likewise, the *(n,k,s)-perceptron convergence theorems* [3] state that a (n,k,s)-perceptron computes a given function $f \in P^n_k$ if and only if f is s-separable. Figure 1 shows an example of 2-separable 4-valued logic function of P^2_5.

Let $V \subset R^n$ with $|V| = v \geq 2$. A *(n,v,s)-partition* is a partition of V by $s \leq v-1$ parallel hyperplanes (namely $(n-1)$-planes) which do not pass through any of the v points. For instance Figure 1 also illustrates an example of $(2,5^2,2)$-partition of the given set of points in P^2_5.

A (n,v,s)-partition determines $s+1$ distinct classes $S_0,\ldots,S_s \subset V$ separated by s parallel $(n-1)$-planes such that $\bigcup_{i=0}^s S_i = V$, $\bigcap_{i=0}^s S_i = \emptyset$ and $S_i \neq \emptyset$. A (n,v,s)-partition corresponds to a s-separable k-valued function $f \in P^n_k$ if and only if all points in the set S_i, have the same value taken out of K. Also, we assume that any two neighboring classes have distinct values. Clearly, the number of associated functions is $k(k-1)^s$.

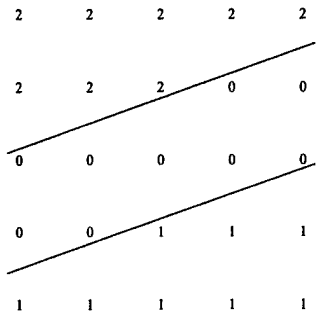


Figure 1: A 2-separable function of P^2_5.

For given $s \geq 2$, the capacity of a (n,k,s)-perceptron with domain V is *approximated* by the product of the number of (n,v,s)-partitions and the number of functions associated with each (n,v,s)-partition of V. Thus counting the number of (n,v,s)-partitions of V is a first step toward calculating the capacity of (n,k,s)-perceptrons.

A *linear partition* of a point set V is a $(n,v,1)$-partition. The enumeration problem for linear partitions is closely related to the efficiency measurement problem for linear discriminant functions in pattern recognition and to many other algorithmic problems. The capacity of $(n,2,1)$-perceptrons with domain V [6] is well-known and is given by

$$|F^n_{2,s}| = \begin{cases} 2^v & \text{if } n \geq v-1 \\ 2^v - 2\sum_{i=n+1}^{v-1} \binom{v-1}{i} & \text{otherwise} \end{cases}$$

[5] estimated lower and upper bounds for the capacity of $(n,2,s)$-perceptrons, using two essentially different enumeration techniques. The paper demonstrated that the exact number of multiple-threshold functions depends strongly on the relative topology of the input set. The results corrected a previously published estimate [7] and indicated that adding threshold levels enhances the capacity more than adding variables.

3 Linear partitions and minimal pairs

An unordered pair $(\vec{x},\vec{y})$ of distinct points of a finite set $V \subseteq R^n$ is a *minimal pair* in V if there does not exist a third point $\vec{z}$ of V which belongs to the open line segment $[\vec{x},\vec{y}]$. A set $V \subset R^n$ is in *general position* if and only if no subset of V of $d+1$ points lies on a $(d-1)$-plane (for $1 \leq d \leq n$) and no two $(n-1)$-planes defined from V are parallel (for $2 \leq d \leq n$).

The number of linear partitions of a finite planar set, no three points of which are collinear, is well known. The number of linear partitions of V in general position is a well known result of pattern recognition [6], and is given by

$$L_{n,v,1} = \sum_{i=0}^{n} \binom{v-1}{i} \quad (1)$$

A relationship between linear partitions and minimal pairs of a finite planar set V is given in [2]. Namely, that the number of linear partitions of V is equal to the number of minimal pairs in V. Moreover, each minimal pair determines two linear partitions of V and conversely, each linear partition of V has exactly two associated minimal pairs [2]. These facts provide an easier way of counting linear partitions for arbitrary finite planar point sets: it suffices to count minimal pairs.

In [2] an explicit formula for the number of linear partitions of the (i,j)-grid (a rectangular part of the infinite grid) is stated, using the correspondence with minimal pairs. The same reference includes an efficient algorithm for counting linear partitions of the (i,j)-grid in linear time with respect to the number of its points.

4 Counting multilinear partitions of subsets in general position

In this section, $V \subset R^n$ is in general position and $L_{n,v,s}$ denotes its number of (n,v,s)-partitions. Clearly, every pair of points in V is a minimal pair. We first obtain the number of $(2,v,s)$-partitions, that is for the two dimensional space, and then we find $L_{n,v,s}$.

We rewrite the number of linear partitions of V in general position with a little modification as

$$L_{n,v,1} = L_{n,v-1,1} + L_{n-1,v-1,1} = \sum_{i=1}^{n} \binom{v-1}{i}$$

The difference with the original formula (1) is that the index i starts with 1. The reason is that we do not include partitions containing empty classes.

To count the $(2,v,s)$-partitions, we associate each $(2,v,s)$-partition with a slope $\sigma \in R$ (or equivalently, a minimal pair) as follows.

- For each of the s partitioning lines, choose one of the two minimal pairs which has the smaller slope;
- The associated slope of the given $(2,v,s)$-partition is the maximum among these smaller slopes.

Lemma 4.1 *The number of $(2,v,s)$-partitions associated with a given slope σ is* $\binom{v-2}{s-1}$.

Theorem 4.1 $L_{2,v,s} \leq \binom{v-2}{s-1}\binom{v}{2}$.

Proof Since V is in general position, then there are $\binom{v}{2}$ slopes. The inequality is explained by the fact that two distinct choices of minimal pairs (or slopes) may have sets of associated $(2,v,s)$-partitions that intersect each other. Therefore a given partition may be counted many times, depending on the configuration of V. For instance consider one of the partitions in Figure 2. Clearly, the same partition can be obtained either by selecting the upper minimal pair or by selecting the lower minimal pair (indeed in this example, they both give the same set of associated partitions, even though the points are in general position). We have equality only when $s = 1$. •

Theorem 4.2

$$L_{n,v,s} \leq \binom{v-2}{s-1} L_{n,v,1} = \binom{v-2}{s-1} \sum_{i=1}^{n} \binom{v-1}{i}.$$

Proof The proof is similar to that of Theorem 4.1. We select a linear partition of V in $L_{n,v,1}$ ways. For each such linear partition, we sort the points of V along the direction of the separating $(n-1)$-plane, that is according to their distance to that hyperplane. To construct a (n,v,s)-partition we must add $s-1$ more parallel $(n-1)$-planes. There are $\binom{v-2}{s-1}$ ways to place $s-1$ new hyperplanes in $v-2$ available positions. Here again we have equality only for $s = 1$. •

Corollary 4.1 *The number of n-input k-valued s-separable functions $f : V \mapsto K$ is*

$$|F_{k,s}^{n}| \leq k(k-1)^s \binom{v-2}{s-1} \sum_{i=1}^{n} \binom{v-1}{i}.$$

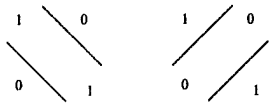


Figure 2: Two $(2,4,2)$-partitions for the same function.

For $k = 2$, Corollary 4.1 gives a much better (tighter) upper bound on the capacity of $(n,2,s)$-perceptrons than the following upper bound obtained in [5].

$$|F_{2,s}^{n}| \leq 2\binom{v-1}{s} \sum_{i=0}^{n-1} \binom{\binom{v}{2}-1}{i} \quad (2)$$

However, in (2), the constant $\begin{pmatrix} v-1 \\ s \end{pmatrix}$ also counts partitions containing empty classes. So, to remove them, we must replace it by $\begin{pmatrix} v-2 \\ s-1 \end{pmatrix}$.

A s-separable function $f : V \mapsto K$ is *permutably homogeneous* if and only if $s \leq k-1$ and, for any partitioning of V, no two separate classes have equal value [3]. These functions are determined by (n, v, s)-partitions in which each of the $s+1$ classes $S_0, \ldots, S_s$ maps to a distinct value in K.

Theorem 4.3 *A permutably homogeneous s-separable function $f : V \mapsto K$ has a unique (n, v, s)-partition.*

Proof Suppose f has two distinct (n, v, s)-partitions P_1 and P_2. Consider two points $\vec{x}$ and $\vec{y}$ which are in the same class with respect to P_1 but in different classes with respect to P_2. We have that $f(\vec{x}) = f(\vec{y})$, because $\vec{x}$ and $\vec{y}$ share the same class in P_1. This means that two distinct classes in P_2 have the same value. This contradicts the definition of permutably homogeneous function. •

Corollary 4.2 *The number of permutably homogeneous n-input k-valued s-separable functions $f : V \mapsto K$ is*

$$|G^n_{k,s}| \leq \frac{k!}{(k-s-1)!} \begin{pmatrix} v-2 \\ s-1 \end{pmatrix} \sum_{i=1}^{n} \begin{pmatrix} v-1 \\ i \end{pmatrix}.$$

5 Counting multilinear partitions of the (k, k)-grid

In this section we generalize the counting method of [2] to enumerate multilinear partitions of the (k, k)-grid K^2.

Let $(\vec{x}, \vec{y})$ be a pair of points from a finite planar grid. Let $a = \max(|x_2 - x_1|, |y_2 - y_1|)$ and $b = \min(|x_2 - x_1|, |y_2 - y_1|)$. One well known condition for a pair $(\vec{x}, \vec{y})$ to be minimal is that a and b are relatively prime, that is their greatest common divisor is 1. $a \perp b$ will denote that the integers a and b do not have common divisor.

Thus the number of minimal pairs corresponds to the number of pairs $(\vec{x}, \vec{y})$ of the (i, j)-grid such that $a \perp b$. Let natural numbers i and j be given so that $i \leq j$. The *generalized Farey (i, j)-sequence* $F_{i,j}$ [2] is the strictly increasing sequence of all the fractions of the form $\frac{b}{a}$, where the integers a and b satisfy: $a \perp b$, $0 < b < a \leq j$, $b \leq i$. Thus the sequence $F_{4,7}$ is as follows:

$$\tfrac{1}{7}\ \tfrac{1}{6}\ \tfrac{1}{5}\ \tfrac{1}{4}\ \tfrac{2}{7}\ \tfrac{1}{3}\ \tfrac{2}{5}\ \tfrac{3}{7}\ \tfrac{1}{2}\ \tfrac{4}{7}\ \tfrac{3}{5}\ \tfrac{2}{3}\ \tfrac{3}{4}\ \tfrac{4}{5}.$$

The *Farey i-sequence* F_i for any positive integer i is the set of irreducible rational numbers $\frac{b}{a}$, with $0 \leq b \leq a \leq i$ and $a \perp b$, arranged in increasing order [2]. So the sequence F_4 is:

$$\tfrac{0}{1}\ \tfrac{1}{4}\ \tfrac{1}{3}\ \tfrac{1}{2}\ \tfrac{2}{3}\ \tfrac{3}{4}\ \tfrac{1}{1}.$$

The length of the sequence $F_{i,j}$ (resp. F_i) will be denoted by $|F_{i,j}|$ (resp. $|F_i|$). Also, $F^d_{i,j}$ (resp. F^d_i) stands for the d-th fraction in $F_{i,j}$ (resp. F_i), $1 \leq d \leq |F_{i,j}|$ (resp. $|F_i|$).

To count the $(2, k^2, s)$-partitions, we associate to each $(2, k^2, s)$-partition a fraction $\frac{b}{a} \in F_{k-1}$ as before Lemma 4.1, where $v = k^2$.

Then to enumerate or generate the $(2, k^2, s)$-partitions associated with a given $\frac{b}{a}$ we will need Lemmas 5.1 and 5.2 below. As in figure 3, rotate a line segment $[\vec{x}, \vec{y}]$ whose slope $\frac{b}{a}$ is irreducible to increase it for a *small* amount, so that we obtain a straight line P with slope $m \notin F_{k-1}$. P is the direction for separation and corresponds to the line segment $[\vec{x}, \vec{y}]$ with slope $\frac{b}{a}$.

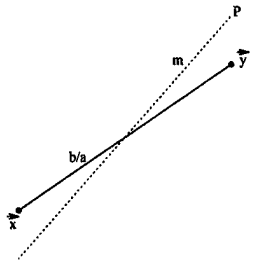


Figure 3: Rotating slope $\frac{b}{a}$ toward slope m.

Lemma 5.1 *Slope $\frac{b}{a}$ of the line segment $[\vec{x}, \vec{y}]$ is greater than or equal to the associated slope of any $(2, k^2, s)$-partition in direction parallel to P, where P is the direction of separation that corresponds to the line segment $[\vec{x}, \vec{y}]$.*

Lemma 5.2 *Slope $\frac{b}{a}$ is equal to the associated slope of a $(2, k^2, s)$-partition in direction parallel to P if and only if at least one of the s separating lines intersects a minimal pair with slope $\frac{b}{a}$.*

[2] obtained an exact formula for the number of linear partitions of the (i, j)-grid. Substituting for the (k, k)-grid we obtain the following corollary.

Corollary 5.1

$$L_{2,k^2,1} = 2k(k-1) + 2(k-1)^2 + 4 \sum_{a \perp b, 0 < b < a < k} (k-a)(k-b).$$

Lemma 5.3 $L_{2,k^2,s} =$

$$4\binom{k^2-1}{s} - 2\binom{k-1}{s} - 2\binom{2k-2}{s} + 4\sum_{a\perp b, 0<b<a<k}\left[\binom{k^2-1}{s} - \binom{ak+bk-ab-1}{s}\right].$$

Proof Let a slope $\frac{b}{a}$ be such that $a \perp b$ (we consider slopes for directions between 0° and 45°. For each such slope $\frac{b}{a}$, Lemmas 5.1 and 5.2 give a simple algorithm to construct (i.e generate) and count the $(2, k^2, s)$-partitions associated with $\frac{b}{a}$.

- Rotate slope $\frac{b}{a}$ to increase it a bit {this gives the direction of an associated $(2, k^2, s)$-partition P};
- Sort the points along this direction;
- Choose s points $\vec{x}_1, \ldots, \vec{x}_s$ out of $k^2 - 1$ points in $\binom{k^2-1}{s}$ ways; {selected points are begining of classes $S_1, \ldots, S_s$ and the point $\vec{x}_0 = (k-1, 0)$ can always be selected for S_0}
- Eliminate all selections of points where no minimal pair of slope $\frac{b}{a}$ is intersected by a separating line of direction P;

To intersect one of the s separating lines, the lower end of a minimal pair with slope $\frac{b}{a}$ must be selected. From Corollary 5.1 we have

- If $b = 0$ then there are $k(k-1)$ lower ends (the number of minimal horizontal segments of the (k,k)-grid). None of them can be selected in $\binom{k^2-1-k(k-1)}{s} = \binom{k-1}{s}$ ways. So the number of ways to select a minimal pair with slope $\frac{0}{1}$ to intersect a separating line is $2\binom{k^2-1}{s} - 2\binom{k-1}{s}$.
- If $a = b = 1$ then there are $(k-1)^2$ lower ends (the number of minimal segments with slope 45°). None of them can be selected in $\binom{k^2-1-(k-1)^2}{s} = \binom{2k-2}{s}$ ways. So the number of ways to select a minimal pair with slope $\frac{1}{1}$ to intersect a separating line is $2\binom{k^2-1}{s} - 2\binom{2k-2}{s}$.
- If $a > b > 0$ then there are $(k-a)(k-b)$ lower ends (the number of horizontal rectangles of size $a \times b$ of the (k,k)-grid). None of them can be selected in $\binom{k^2-1-(k-a)(k-b)}{s} = \binom{ak+bk-ab-1}{s}$ ways. So the number of ways to select a minimal pair with slope $0 < \frac{b}{a} < 1$ to intersect a separating line is $4\sum_{a\perp b, 0<b<a<k}\left[\binom{k^2-1}{s} - \binom{ak+bk-ab-1}{s}\right]$.

Taking the total sum of all three cases yields the formula. This completes the proof. •

Theorem 5.1

$$L_{2,k^2,s} = 4\binom{k^2-1}{s}|F_{k-1,k-1}| - 4\binom{k^2-1}{s} - 2\binom{k-1}{s} - 2\binom{2k-2}{s} - 4\sum_{a\perp b, 0<b<a<k}\binom{ak+bk-ab-1}{s}.$$

Proof Follows from Lemma 5.3 and the fact that the last sum is over $|F_{k-1,k-1}|$. •

Lemma 5.4 $ak + bk - ab - 1 < k^2 - 1$.

Corollaries 5.2 and 5.3 below give, respectively, a lower bound and an upper bound on the number of $(2, k^2, s)$-partitions of the (k,k)-grid.

Corollary 5.2 $L_{2,k^2,s} >$

$$4\binom{k^2-1}{s}|F_{k-1,k-1}| - 2\binom{k-1}{s} - 2\binom{2k-2}{s}.$$

Corollary 5.3 $L_{2,k^2,s} <$

$$4\binom{k^2-1}{s}(|F_{k-1,k-1}|+1) - 2\binom{k-1}{s} - 2\binom{2k-2}{s}.$$

The asymptotic formula for the length of the generalized *Farey* (i,j)-sequence (for $i \le j$) is given in [2] as $|F_{i,j}| = \frac{3i^2j^2}{\pi^2} + O(i^2 j \log j) + O(ij^2 \log\log j)$. Hence, substituting for the (k,k)-grid we obtain $|F_{k-1,k-1}| = \frac{3k^4}{\pi^2} + O(k^3 \log k) + O(k^3 \log\log k)$.

Theorem 5.2 $L_{2,k^2,s} \approx$

$$4\binom{k^2-1}{s}\frac{3k^4}{\pi^2} - 2\binom{k-1}{s} - 2\binom{2k-2}{s} + 2\binom{k^2-1}{s}.$$

Proof We take the mean of the lower and upper bounds and replace $|F_{k-1,k-1}|$ by its formula given above. •

Corollary 5.4 *The number of 2-input k-valued s-separable logic functions is* $|F^2_{k,s}| < k(k-1)^s L_{2,k^2,s}$.

The example given in figure 2 explains the inequality in Corollary 5.4. For instance, the XOR funcion has exactly two $(2,4,2)$-partitions. For $s = 1$ we obtain an asymptotic formula.

Corollary 5.5 *The number of permutably homogeneous 2-input k-valued s-separable logic functions is* $|G^2_{k,s}| \approx \frac{k!}{(k-s-1)!} L_{2,k^2,s}$.

6 Complexity of counting the number of $(2, k^2, s)$-partitions

Given a point set V in the plane, there are $\frac{v^2-v}{2}$ pairs of points. Generally speaking, each pair requires $v-2$ tests to check whether it is minimal. Thus the complexity of the general case for the extraction of all minimal pairs is $O(v^3)$. However, if the considered $v = ij$ points are all the points of the (i,j)-grid, then this algorithm turns out to be linear with respect to v, namely $O(v)$, on the basis of Corollary 5.1 and the fact that the successor of a member of $F_{i,j}$ can be calculated in a constant time [2]. The simple linear time algorithm for generating $F_{i,j}$ is described in [2].

In Figure 4 we show the fast algorithm for counting $(2, k^2, s)$-partitions of the (k,k)-grid. It is a modified version of the [2] algorithm for enumerating the linear partitions of the (i,j)-grid, so, the reader is refeered to [2] for a complete proof of correctness of the algorithm. The modifications include replacing the (i,j)-grid by the (k,k)-grid and computing the binomial coefficients present in Lemma 5.3.

```
Read k and s;
d := 1; c := k − 1;
b := 1; a := k − 2;
L_{2,k^2,s} := 12 C(k^2 − 1, s) − 2 C(k − 1, s) − 2 C(2k − 2, s) −
               4 C(k^2 − k, s) − 4 C(k^2 − 2k + 1, s);
Repeat
   j := d; i := c;
   d := b; c := a;
   r := ⌊(k − 1 + i)/c⌋;
   b := rd − j;
   a := rc − i;
   L_{2,k^2,s} := L_{2,k^2,s} + 4 C(k^2 − 1, s) − 4 C(ak + bk − ab − 1, s);
Until b = k − 2 and a = k − 1;
Write out L_{2,k^2,s};
```

Figure 4: Fast algorithm for counting $(2, k^2, s)$-partitions.

Let $\frac{i}{i}$ and $\frac{d}{c}$ be two consecutive elements of $F_{k-1,k-1}$. It has been shown in [2] that the immediate successor $\frac{b}{a}$ of $\frac{d}{c}$ is always determined by the relations $b = rd - j$ and $a = rc - i$, where $r = \lfloor \frac{k-1+i}{c} \rfloor$. This implies that the computation of a consecutive member of $F_{k-1,k-1}$ can be completed in a constant time. Another consequence of this fact is that initialization is easy: we can always start with the first two *Farey* numbers $F^1_{k-1,k-1} = \frac{1}{k-1}$ and $F^2_{k-1,k-1} = \frac{1}{k-2}$ and then generate the whole $F_{k-1,k-1}$ sequence in a loop using the above relations.

The last sum of the formula given in Lemma 5.3 necessarily reduces to the sum over the *Farey* $(k-1, k-1)$-sequence. Suppose that k^2 and s require each $O(\log k^2)$ and $O(\log s)$ bits for memory storage, then the complexity of computing the binomial coefficients is $O(s)$. Therefore, for each generation of a number $\frac{b}{a} \in F_{k-1,k-1}$, it takes $O(s)$ to compute the number of $(2, k^2, s)$-partitions associated with $\frac{b}{a}$. Since there are k^2 elements in the (k,k)-grid, then it takes $O(sk^2) \leq O(k^4)$ time to calculate $L_{2,k^2,s}$. Clearly, the time complexity is polynomial on k.

7 Conclusion and further research

In this paper we have introduced the concepts of multilinear separability and multilinear partitions, and derived formulae for the number of multilinear partitions of subsets in general position and of the (k,k)-grids. A more difficult problem would be to enumerate partitions for higher dimension grids. One interesting open question is the following. In how many ways can we partition a finite set $V \subset R^n$ using s *non-necessarily parallel* hyperplanes? The answer to this question gives (bounds on) the capacity of some classes of neural networks.

References

[1] M.H. Abd-El-Barr, S.G. Zaky and Z.G. Vranesic (1986), *Synthesis of multivalued multithreshold functions for CCD implementation*, IEEE Trans. Comp., V.C-35, N.2, February, pp.124-133.

[2] D. Acketa and Joviša Žunić (1991), *On the number of linear partitions of the (m,n)-grid*, Inf. Proc. Let., North-Holland, V.38, pp.163-168.

[3] A. Ngom, C. Reischer, D.A. Simovici and I. Stojmenović (1998), *Learning with permutably homogeneous multiple-valued multiple-threshold perceptrons*, Proc. 28th IEEE ISMVL, pp.161-166.

[4] Z. Obradović and I. Parberry (1992), *Computing with discrete multivalued neurons*, Jour. Comp. Sys. Sci., V.45, N.3, pp.471-492.

[5] S. Olafsson and Y.A. Abu-Mostafa (1988), *The capacity of multilevel threshold functions*, IEEE Trans. PAMI, V.10, N.2, pp.277-281.

[6] K.-Y. Siu, V. Roychowdhury and T. Kailath (1995), *Discrete neural computation: A theoretical foundation*, Prentice Hall Inf. Sys. Sci. Ser., Thomas Kailath, Series editor.

[7] R. Takiyama (1985), *The separating capacity of a multithreshold threshold element*, IEEE Trans. PAMI, V.7, January, pp.112-116.

B-ternary Logic Based Asynchronous Micropipeline

Yasunori Nagata
University of the Ryukyus
Dept.of Elec'l & Elec's Eng.
1 Senbaru Nishihara
Okinawa, 903-0213 JAPAN
ngt@eee.u-ryukyu.ac.jp

D. Michael Miller
University of Victoria
Dept. of Computer Science
PO Box 3055, Victoria,B.C.
Canada V8W 3P6
mmiller@csr.uvic.ca

Masao Mukaidono
Meiji University
Dept. of Information Science
Tama-ku Kawasaki-shi,
214-0038 Japan
masao@cs.meiji.ac.jp

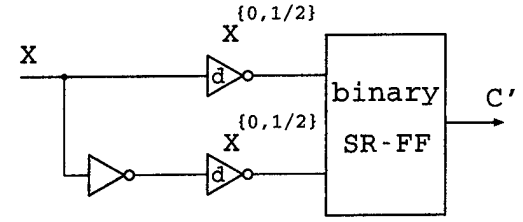
Abstract

In this paper, a B-ternary logic based asynchronous pipeline is presented. The pipeline processes binary data elastically. It has high speed operation potential in spite of having an idle phase, because the stages of the pipeline operate concurrently. The mechanism for correct pipeline behavior and the designed circuits are provided.

1. Introduction

Pipeline architecture is widely used for high-speed computations, for instance, image processing, arithmetic units or instruction decoding within computers. Data in a pipeline system are concurrently processed at successive stages. Thus, for this feature, memory elements and processing logic circuits appear alternately. Originally, an asynchronous pipeline was designed by I. E. Sutherland [1]. It has been called an event driven *micropipeline*. Pipelines can of course be implemented using either clock-controlled or Request/Acknowledge controlled circuitry. Some forms of pipeline are inelastic and look like shift-registers. In an elastic pipeline, not only is the amount of data variable but so is the input/output speed. The pipeline behaves as a first-in first-out queue (FIFO). Asynchronous processing schemes possess some significant advantages for such a pipeline. These include simpler design results and circuitry suited to implementing elastic systems. In contrast, it was shown that the implementation of the elastic pipeline as a synchronous system is significantly more complicated [1].

In this paper, a B-ternary logic [2] based asynchronous pipeline is considered. The pipeline presented processes binary data elastically. It has high speed operation potential, in spite of having an idle phase, because the stages of the pipeline operate concurrently. To implement the design, we employ ternary AND, OR, NOT gates and C-elements [5]. The operations of AND, OR and NOT are min, max and complement, respectively. (For details of the CMOS implementation of these elements, refer to [5].) The ternary C-element is a binary memory element that accepts ternary input signals. Figure 1 shows the ternary-in, binary-out, Muller C-element, which contains a kind of inverter $X^{\{0,1/2\}}$. $X^{\{0,1/2\}}$ is said to be a *down-literal* d_2. It performs as $X^{\{0,1/2\}} = \begin{cases} 1: & X = 0 \text{ or } 1/2 \\ 0: & X = 1 \end{cases}$. The C-element is set to 1 when its input is 1, it resets to 0 when the input is 0 and it captures the previous data value when the input is 1/2. In practice, the value 1/2 is filtered out by means of a threshold in the down-literals. This also avoids metastability in the binary SR-FF of Figure 1.



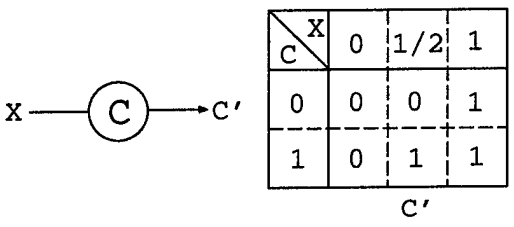


Figure 1. Ternary-in binary-out C-element. upper: circuit lower left: symbol lower right: truth table.

The C-element is useful in data paths of asyn-

0-7695-0161-3/99 $10.00 © 1999 IEEE

chronous systems to separate the stages in a data stream. We exploit a three state coding which has three detectable states: logical 0, logical 1 and a null state (spacer) 1/2. In a data stream, bits of data (0 or 1) and spacers (1/2) appear alternately to separate successive data bits. This three state coding is also referred to as 2-phase data transferring.

The advantages of the B-ternary logic based asynchronous system are as follows.

1. It reduces the interconnection complexity in an asynchronous system (almost twice the connections of the synchronous case) [3][4][5].
2. The number of transistors in ternary circuits (gates) is nearly equal that of 2-rail logic. Note that the binary asynchronous system based on 2-rail logic is equivalent to the B-ternary system [5].
3. B-ternary circuits can be directly connected to binary ones without data encoding/decoding.
4. Logic circuits for the purpose of data processing can be designed using CAD tools intended for binary synchronous circuits (e.g. espresso).
5. The presented ternary-in binary-out memory (Muller C-element) avoids metastability.

In the following section, the basis of a micropipeline (simply pipeline) and a design of a B-ternary asynchronous pipeline/FIFO are explained.

2. Pipeline

A block diagram of the general pipeline is given in Figure 2. As shown, memory elements (Mem.) appear between the processing logic circuits (L.C. : combinational circuit) throughout the pipeline. The controller connected to each memory determines whether the data is stored in the memory or not (retain). Once data is stored it is processed by the next L.C. and the result will be stored in the next memory element. Therefore, in the case of the asynchronous pipeline, the memory controllers resolve the timing of data transfers through the pipeline.

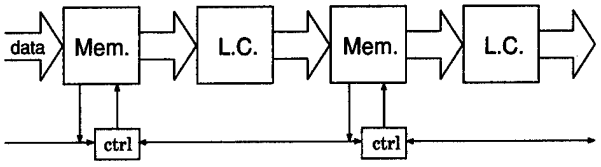


Figure 2. Generalized micropipeline.

The behavior of the asynchronous pipeline is illustrated in Figure 3. The m1~m7 are the states of the pipeline at certain moments in time, and the #1~#4 are the stages of the circuit. State transitions of m1~m5 show data transfers: #2→#3, #1→#2, #4→out, and #3→#4, respectively. The transition m5→m6 indicates a concurrent data transfer of #2→#3 and #4→out. Finally, the data in #3 moves to #4 during m6→m7.

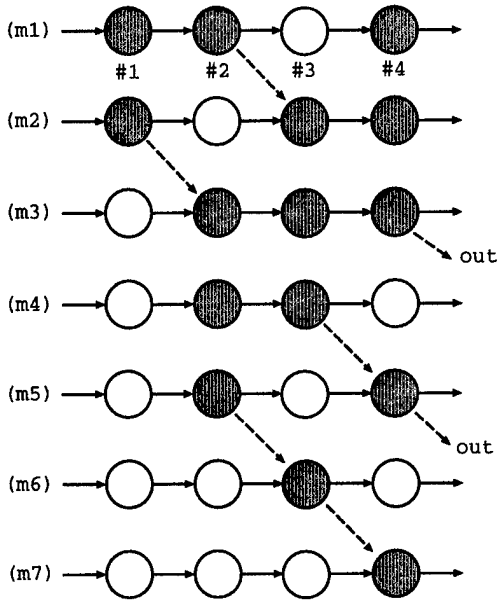


Figure 3. Data transfer examples in an elastic pipeline.

To summarize the above considerations,

(a1) data transfer between successive stages progresses sequentially, or

(a2) data transfers in two or more not necessarily adjacent pairs of stages can progress concurrently.

Thus, from Figure 2 and 3, we see that each datum can move to the next place if it is empty, and moreover, these transfers can be done in parallel on the above conditions *(a1)* and *(a2)*.

The circuit that consists of only memories and controllers, omitting L.C.'s, is an elastic FIFO. In the following, since no generality will be lost, we use a FIFO rather than a general asynchronous pipeline.

3. Design of a FIFO and its sub-circuits

In this section, design of a FIFO and its sub-circuits are explained. Figure 4 illustrates the generalized ternary asynchronous FIFO. The FIFO consists of

ternary-in binary-out Muller C-elements, request (Req) circuits, and Controllers (ctrl's).

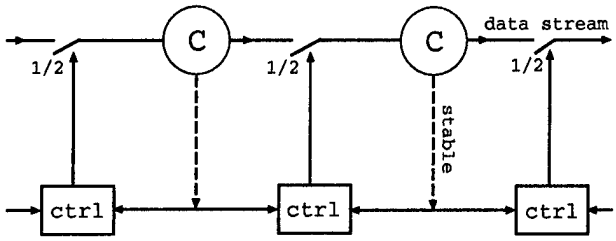


Figure 4. Generalized simple ternary asynchronous FIFO.

For the FIFO of Figure 4 to operate correctly, the following conditions are required:

(1) If both of the following hold, data transmission can be performed.

(a) The memory (i.e. C-element) of a certain stage has data and it is stable.

(b) The memory of the next stage of (a) is empty.

(2) If either (a) or (b) (or both) does not hold, then the memory keeps the current data.

Note that *memory empty* means that a C-element has transmitted its current data to the next C-element and the next C-element has stabilized. Under these conditions, we are able to have two types of behavior:

(i) sequential behavior of data transmission and wiping (with 1/2) in the data path,

(ii) concurrent behavior of data transmission and wiping (with 1/2) in the data path.

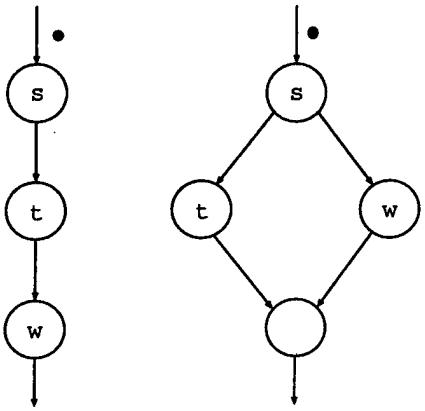


Figure 5. Sequential (left) and concurrent (right) behaviors of data transmission and wiping with 1/2.

Each behavior can be expressed by a signal transition graph (STG) as shown in Figure 5, where we use the symbols: **t**: Req to transfer data, **w**: wiping with 1/2, and **s**: C-element stabilized.

To achieve the above conditions with B-ternary logic circuits, we need the following sub-designs.

(A) Ternary-in binary-out Muller C-element with stable-state detector.

Under the unbounded delay model, it is not sufficient to consider the signal on the input-line of the memory to know whether the memory has stored the signal value or not. Due to gate delays, there are some possibilities of progression to next behavior before the C-element stabilizes. Thus we combine the C-element with a *coincident circuit* to detect a stable memory state as shown in Figure 6. The circuit outputs: 0 when the value in memory is changing; 1 when there is no change or the value has changed completely; 1/2 when the input-signal is 1/2.

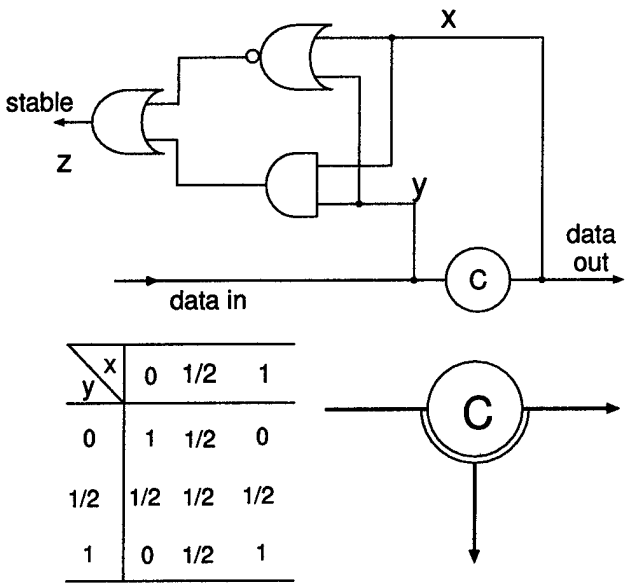


Figure 6. C-element with ternary coincident circuit.

(B) Data-gating circuit by Req-signal.

The data-gating circuit is located at the output side of memory. The gate opens to transmit stored data from the preceding C-element to the next, or to hold the data back while transmitting 1/2 instead. Let the data-bit be a, the circuit can be expressed with restricted Req signal $x \in \{1/2, 1\}$ as follows,

$$g = a \cdot x \vee \sim x \qquad (1)$$

where "$\cdot$", "$\vee$" and "$\sim$" indicate ternary min, max and complement operations, respectively. The circuit for

this equation is shown in Figure 7. It outputs 1/2 when Req signal $x = 1/2$ or it outputs data a when $x = 1$.

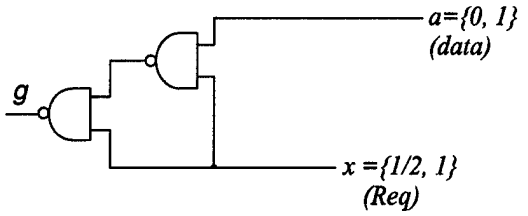


Figure 7. Restricted data-gating circuit.

(C) Data empty detection circuit.

To make a memory empty and to detect it, two types of behavior were mentioned above: sequential and concurrent data transfer (**t**) and wiping a data-path with 1/2 (**w**) on the memory's stable state.

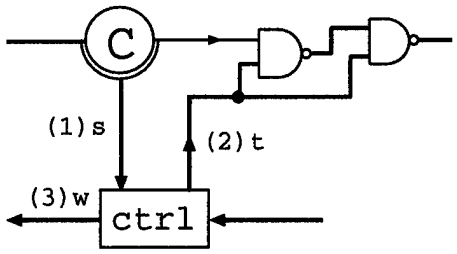


Figure 8. Circuitry for sequential empty behavior.

Figure 8 shows conceptual circuitry for sequential empty behavior such as the STG in Figure 5 (left). In the circuit, the controller detects the stable state of the memory (C-element) and then controls the sequential events **t** and **w**.

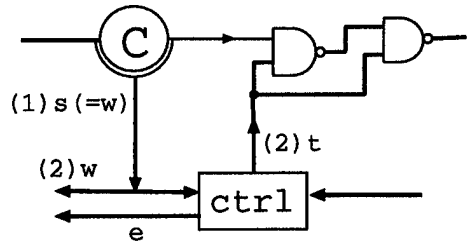


Figure 9. Circuitry for concurrent empty behavior.

In Figure 9, for concurrent **t** and **w** behavior on **s**, we need one more interconnection between the controllers. The design detail for these behaviors is provided in the next section.

4. B-ternary logic based pipeline

For a B-ternary micropipeline, we designed two types of FIFO as shown in Figures 10 and 11. These two micropipelines (FIFOs) satisfy the desired conditions *(a1)* and *(a2)* mentioned in section 2. Namely, these two micropipeline can perform concurrent data processing. However, they have different procedures to "empty" each memory. Figure 10 shows two stages of a FIFO with the sequential data empty circuit. The operations of this FIFO is as follows.

1. When data has reached a memory (C-element) and the memory has stabilized, this condition is indicated by a specific controller ctrl.

2. If both the stabilized signal and a memory empty signal from the next stage meet, the ctrl opens the gate by Req signal (t) such as $1/2 \rightarrow 1$, and the data in the memory will move to the next memory element.

3. When both the memory-stable signal (s) and a data transfer signal (t") meet at the AND gate, it properly is an "empty signal".

In contrast, the FIFO in Figure 11 has concurrent data empty connections. It behaves as follows.

1. A memory generates its stable-state signal.

2. • If both the stable-state signal and empty signal from the next memory are indicated, ctrl opens the gate to send data.
   • Concurrently, the preceding ctrl closes the data path to the memory, that is, wiping the path by 1/2.

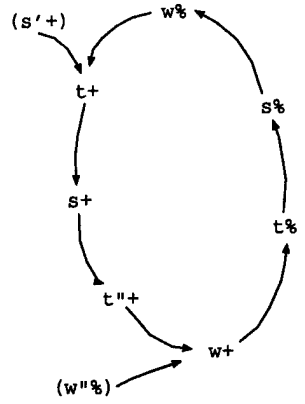


Figure 12. STG for the pipeline in Figure 10.

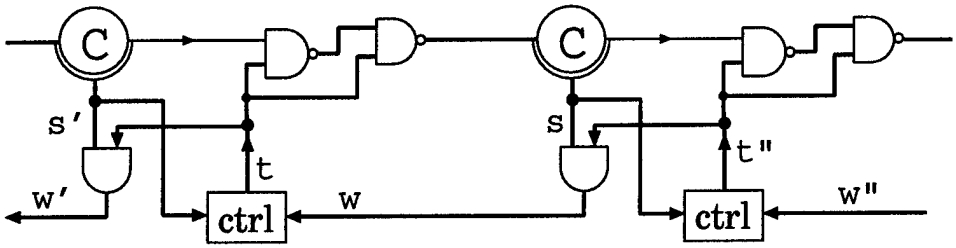
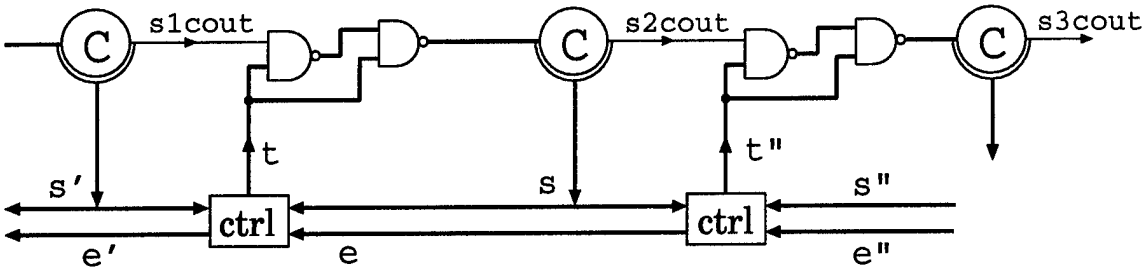


Figure 10. FIFO (pipeline) with sequential empty procedure.



centerline

Figure 11. FIFO (pipeline) with concurrent empty procedure.

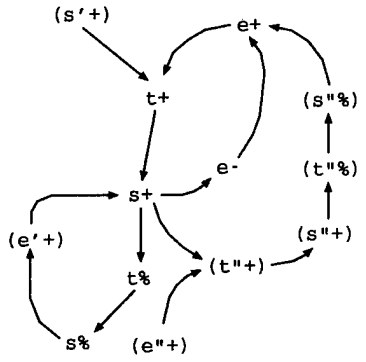


Figure 13. STG for the pipeline in Figure 11.

Signal transition graphs (STGs) of these two kinds of pipeline are shown in Figure 12 and 13, respectively. Note that these STGs are illustrated for the t - s - w (or e) loop in Figure 10 (or Figure 11), and signal transitions in parentheses are for the surrounding loops' behaviors.

From these STGs, we can design ctrl's (controllers) as shown in Figure 14 and Figure 15. These ctrl's are almost the same as those presented in the literature [5]. In Figure 14, the down-literal d_1 operates as $X^{\{0\}} = \begin{cases} 1: & X = 0 \\ 0: & X = 1/2 \text{ or } 1 \end{cases}$ Refer to section 1 for d_2.

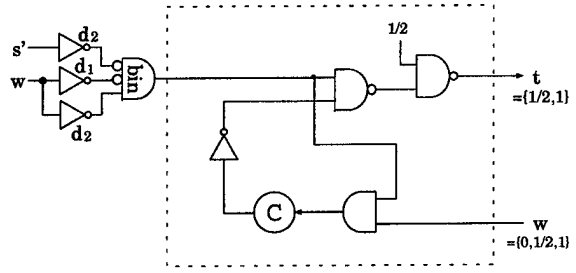


Figure 14. Ctrl for the pipeline in Fig.10.

The pipeline in Figure 11 needs one extra interconnection between ctrl's to achieve concurrent behavior, to wipe the data-path and enable the next data transfer.

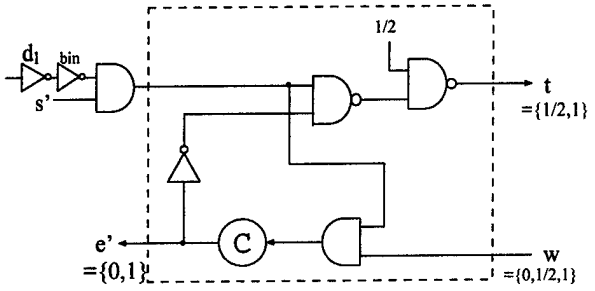


Figure 15. Ctrl for the pipeline in Fig.11.

In the ctrl in Figure 15, the signal e' signals the empty state to the previous (i.e. sending) ctrl. One can see that both ctrl's are quite simple.

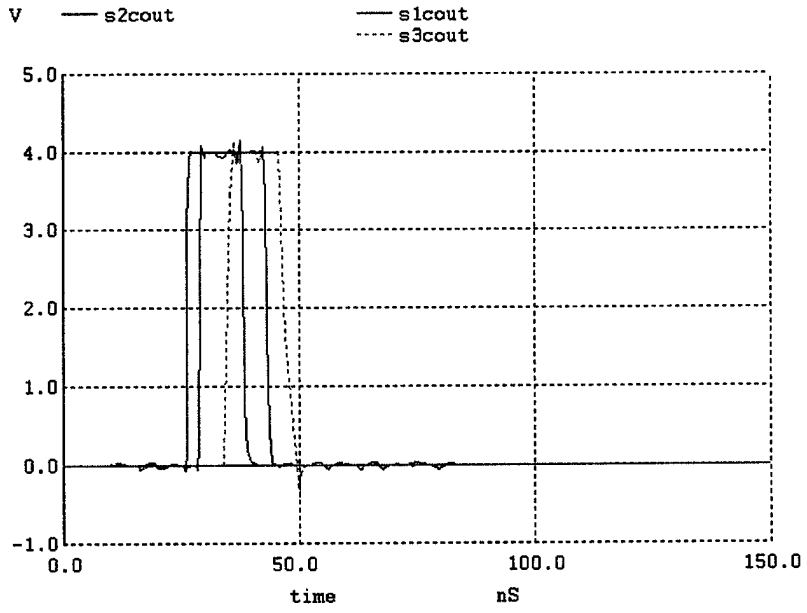


Figure 16. SPICE simulation of the circuit in Figure 11.

Figure 16 is the result of a SPICE simulation of the FIFO with concurrent empty procedure (Figure 11). Note that the three pulses in the figure are at the outputs of the three C-elements. The figure shows a propagation of a pulse through the three C-elements.

5. Conclusion

We considered the behaviour of an elastic pipeline exploiting asynchronous systems. To achieve the desired system behavior, we designed two kinds of micropipeline. Interestingly, these different behavioral pipelines can be controlled in almost the same manner. We stress that ternary asynchronous micropipelines have attractive advantages; the system processes binary data elastically, and they have high speed operation potential in spite of having an idle phase, because the stages of the pipeline operate concurrently. Finally, we found that the STG of a micropipeline is relatively complicated due to the interaction of behavior between adjacent stages.

References

[1] I. Sutherland, "Micropipeline," *Commun. ACM,* Vol.32, pp.720-738, June 1989.

[2] M. Mukaidono, "Regular Ternary Logic Functions – Ternary Logic Functions Suitable for Treating Ambiguity," *IEEE Trans. Comput.,* Vol. c-35, No.2, pp.179-183, Feb. 1986.

[3] R. Mariani et al., "A Useful Application of CMOS Ternary Logic to the Realization of Asynchronous Circuits," *Proc. 27th ISMVL,* pp.203-208, May 1997.

[4] T. Hanyu, T. Saito and M. Kameyama, "Asynchronous Multiple-Valued VLSI System Based on Dual-Rail Current Mode Differential Logic," *Proc. 28th ISMVL,* pp.134-139, May 1998.

[5] Y. Nagata and M. Mukaidono, "Design of an Asynchronous Digital System with B-Ternary Logic," *Proc. 27th ISMVL,* pp.265-271, May 1997.

State Assignment Techniques in Multiple-Valued Logic

KJ Adams*, JG Campbell*, LP Maguire#, JAC Webb^

*School of Computing and Mathematical Sciences,
University of Ulster, Magee College, Londonderry,
BT48 7JL, Northern Ireland, UK
Email kj.Adams@ulst.ac.uk

#Intelligent Systems Engineering Lab.
Faculty of Engineering, University of Ulster
Magee College, Londonderry,
Northern Ireland, UK

^School of Electrical and Mechanical Engineering,
University of Ulster, Jordanstown,
Northern Ireland, UK

Abstract

Multiple-Valued Logic (MVL) functions are implemented via Boolean multiple-wire arrangements where a careful state assignment methodology is used to ensure efficient implementation regimes. A 'power of N' module is proposed for GF (2^3). The method avoids the need to factorize the polynomial and circuits can be realised using a combination of NOT AND and XOR functions. In addition, a novel transform over GF (2^2) is proposed which shows promise when compared to the Reed-Muller-Fourier transform, in its capacity to produce zero co-efficients. A possible implementation strategy, using Field Programmable Gate Arrays (FPGAs) is briefly discussed.

1. Introduction

State assignment techniques provide a way of implementing multiple-valued logic functions with Boolean variables. For example Menger [1] mapped the elements of GF(2^3) to the set of Boolean vectors with three co-ordinates. As each co-ordinate is a logic 0 or logic 1 there are eight possible states so such a mapping is one to one and onto. PLUS and TIMES modules were designed with Boolean algebra. The PLUS module used three inputs to carry the state assignment for the first element to be added and another three inputs for the second. The three outputs carried the state assignment belonging to their sum. The TIMES module worked similarly. With these two modules any function defined over the elements of GF(2^3) into the same elements can be implemented.

State assignment techniques are currently used for Finite State Machines [2], where the nodes are embedded into a Boolean N-cube. To aim for low power consumption nodes that frequently switch between each other are embedded so that the hamming distance between their assignments is as small as possible. In [2] the authors report that the process of state embedding can be time consuming since state embedding is an NP-complete problem. Thus they used an optimal heuristic in their algorithm. Our approach to finding good assignments has been heuristic rather than exhaustive because of the large number of possibilities. For instance there are 40320 possible state assignments for GF(2^3).

Existing transform techniques for implementing multiple-valued logic have draw backs. In [3] a hybrid transform is proposed that can represent a quaternary function by a polynomial where powers are raised within GF(2^2) but standard algebra with real coefficients is then used for the rest of the computation. The authors proposed a universal logic module but regarded it as too computationally expensive for implementation. In [4] and [5] the Reed-Muller-Fourier transform is introduced which uses the 4EXP and 4AND functions to generate the basis for the transform. Then mod4 arithmetic is used with the coefficients. The advantages of RMF are that the number of coefficients needed could be smaller than in

0-7695-0161-3/99 $10.00 © 1999 IEEE

Galois field representation, they are also easily calculated [5]. Their disadvantage is lack of circuit implementation of the multiplication and addition operations [3]. Current Mode CMOS circuits have been used in [6] to implement the field operations over $GF(2^2)$ and $GF(2^4)$ The circuits for $GF(2^2)$ uses four possible current levels on a single wire. Their $GF(2^4)$ circuits required two wires. For large fields using a bundle of wires to carry the representation may be more practical than packing a lot of separate signal levels onto the same wire. Thus state assignment techniques will always come under consideration.

In section two of this paper it is demonstrated that for any transform if the matrix representation of the basis is invertible over a suitable integer ring then we can use integer coefficients from that ring and ring arithmetic for the computation. This leads to a hybrid transform over $GF(2^2)$ similar to that of [3] except that mod4 arithmetic is used instead of real arithmetic. The efficiency of this new hybrid, in terms of zero coefficients, is comparable to the RMF.

This paper offers a way of implementing multiple-valued logic using state assignments that permit the use of simpler circuit designs. Thus tackles the problems mentioned above. An assignment for $GF(2^3)$ leads to the design of a new power of 'N' module. This eliminates the need for repeated multiplication through cascaded modules just for raising powers. An assignment for $GF(2^2)$ is proposed that uses just one AND gate to generate the transform matrix. Then the inverse is taken using mod4 arithmetic. Which is useful because a mod4 adder is just two binary stages ignoring the carry from the second stage.

2. Transforms using integer rings

In order for a square matrix to be invertible in an integer ring it is necessary and sufficient that the determinant of the matrix be a unit in the ring [7]. For an integer ring *Z/nZ* the units are those integers that are relatively prime for n [8]. For example the powers of a variable x in $GF(2^2)$ has the following matrix,

Table 1. Powers of x in $GF(2^2)$

1	x	x^2	x^3
1	0	0	0
1	1	1	1
1	2	3	1
1	3	2	1

As this matrix has determinant equal to one mod4 a mod4 inverse exists which is as follows,

Table 2. Inverse mod4 Hybrid Transform

1	0	0	0
0	1	1	2
0	1	2	1
3	3	1	1

The existence of this hybrid means that functions of $GF(2^2)$ into $GF(2^2)$ can be realised by a hybrid polynomial where powers are raised in $GF(2^2)$ algebra but the rest of the calculation uses mod4 arithmetic.

As an illustration f(x) has vector representation $(2,3,1,0)^T$ if we multiply this vector by the matrix of Table 2 we obtain the coefficients $(2,0,1,0)^T$. This means that $f(x) = 2 + x^2 \pmod 4$.

All 256 single variable functions over $GF(2^2)$ were examined and the least number of coefficients needed for each function was calculated, results being optimised from the 4 polarities available in GF(4). An expected frequency of 2.301 coefficients was obtained, using the results from [11] the expected frequencies for RMF is 2.316 and the GF method had 2.332.

3. An assignment of binary triples to the elements of $GF(2^3)$

The motivation behind this assignment was to examine the properties of Boolean algebra and $GF(2^3)$ algebra and aim to use the assignment to make operating in $GF(2^3)$ as similar to working in Boolean algebra as possible.

The zero element e_0 of $GF(2^3)$ has the property that, $e_0+e_i = e_i$ and $e_0*e_i = e_0$, also, using + as the co-ordinate wise exclusive-or operation $(0, 0, 0) + (x_1, x_2, x_3) = (x_1, x_2, x_3)$, likewise $(0, 0, 0)$ **AND** $(x_1, x_2, x_3) = (0, 0, 0)$. So we let e_0 be assigned to (0, 0, 0). As $e_0*e_1 = e_1$ and (1, 1, 1) **AND** $(x_1, x_2, x_3)=(x_1, x_2, x_3)$ we assign e_1 to (1, 1, 1).

In Boolean triples let $X = (x_1, x_2, x_3)$, then the complement of X is $(x_1, x_2, x_3)+(1, 1, 1)$. In $GF(2^3)$ $e_3+e_2 = e_1$, $e_5 + e_4 = e_1$, $e_7 + e_6 = e_1$, and $e_7 + e_6 = e_1$. So if we arrange that e_3 is assigned to (1, 0, 0), e_5 to (0, 0, 1) and e_7 to (0, 1, 0). Then to preserve this property with exclusive-OR we must also assign e_2 to (0, 1, 1), e_4 to (1, 1, 0) and e_6 to (1, 0, 1). This completes the assignment. (See Table 3)

With this assignment the Boolean equivalent of multiplication becomes :

$(x_1,x_2,x_3)*(y_1,y_2,y_3)$ =

$y_3x_1+(y_1+y_2)x_3+(y_2+y_3)x_2$,

$y_1x_2+(y_2+y_3)x_1+(y_1+y_3)x_3$,

$y_1x_2+(y_1+y_2)x_1+(y_1+y_3)x_2$

The logic circuits for multiplication require six input wires and three output wires. The logic needed for each output was worked out separately using a Karnaugh maps. For each fixed set of values of the Y vector the multiplication becomes a function of X only. These functions were simplified in a three variable Karnaugh map. Then each function of X was multiplied by the mintern that represented the associated Y value. Finally all eight results were added using the exclusive-OR operation and further simplified.

If a particular polynomial is to be realized by Boolean modules the X input (x_1,x_2,x_3) can be the variable and the Y input can represent the constant by which it is multiplied. If we allow the Y input to vary over it's range we obtain Table 6. Inspecting the table we see that all the co-ordinates are **exclusive-OR** of the various X terms. This leads to a simple constant times X module which consists of just two **exclusive-OR** gates and a strap.

Also when realizing Galois polynomials it would be useful to have at hand the X to the power of N function. The derivation of each co-ordinate raised to the power of N is given in Table 7.

Inspecting the table reveals that the power of N module can be realized by three copies of the function $f(x_i,x_j,x_k) = x_i+x_j+(x_jx_k)$

For a practical example we consider a function first used by Menger [1] which is,

$G(X) = e_5 + e_6x + e_3x^6$. This factorizes to,

$$G(X) = [e_3(X+e_1)(X+e_5)(X+e_4)^2+e_4](X+e_1)(X+e_5).$$

Which can be realized with four plus modules and five times modules. Menger used the assignment given in Table 4. From which he designed a times module using 9 **AND** gates and 8 **exclusive-OR** gates. The plus module consists of only three **exclusive-or** gates. Which means that his realization uses a total of 52 **exclusive-OR** gates and 45 **AND** gates. A times module for the assignment of Table 3 would use 9 **AND** gates and 9 **exclusive-OR** gates.

Realising the function directly with the power of N modules yields the network of Fig 1 which uses 10 **exclusive-OR** gates, 3 **AND** gates and 4 **NOT** gates. The technique used means that the polynomial did not have to be factorized and the use of a times module just for multiplying by constants was no longer needed. Indeed we never used a times module at all since we never multiplied by a completely unknown variable.

Designing a constant times X module for Menger's assignment would need 3 **exclusive-OR** gates compared to our two. Menger's assignment also leads to a more complicated X to the power of N function which does not have the standard form of $f(x_i,x_j,x_k)= x_i+x_j+(x_jx_k)$.

For example with Menger's assignment,

$x^3 = (x_1 + x_2x_3, x_1 + x_2 + x_3 + x_1x_3, x_1x_2 + x_3 + x_2x_3)$

4. A method of implementing two variable functions over GF(2^2)

In this section we develop a method of implementing two variable functions over GF(2^2) with two wires, each wire either carries a logic 1 or a logic 0. We use the natural binary assignment e_0 to (0,0), e_1 to (0,1), e_2 to (1,0) and e_3 to (1,1). Using the arithmetic basis of [9] leads to the following matrix representation of the basis:

$$L = \begin{matrix} 1 & x_1 & x_2 & x_1x_2 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{matrix}$$

This matrix has an inverse over the ring of integers mod 4 which is:

$$\begin{matrix} 1 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 \\ 3 & 0 & 1 & 0 \\ 1 & 3 & 3 & 1 \end{matrix}$$

So in line with the ideas presented in section one a hybrid transform exists where any quaternary function

can be realized as a linear combination of these functions with coefficients and calculations taken mod4.

For two variables we can take the Kronecker product of L with itself and use four wires. The inverse for this will be the Kronecker square of the inverse of L. As an example we will again implement the two variable function given in [3] by Falkowski and Rahardja. In Table 5 we rewrite the function with two variables z_1 and z_2 where x_1, x_2 are the wires assigned to z_1; x_3,x_4 to z_2. Our method leads to the following implementation:

$$F(z_1, z_2) = F(x_1, x_2, x_3, x_4) = 3x_1 + x_2 + x_1x_2 + x_3 + 3x_1x_3 + 2x_1x_2x_3 + 2x_4 + x_1x_4 + x_1x_2x_4 + 2x_3x_4 + 2x_2x_3x_4 + 2x_1x_2x_3x_4$$

where for example $2x_2x_3x_4$ is worked out by the logical **AND** of x_2 and x_3 and x_4 then the result which is either a 1 or a 0 is multiplied by 2. All additions are carried out mod 4, where a mod 4 adder is simply a two stage binary adder where we can ignore the carry digit from the second stage.

5. A new set of basis functions for GF(2^2)

Considering the matrix L of the previous section, as this matrix and its inverse has a triangular structure, we can consider an extended set of 24 polarities obtained by permutating the rows of L in a similar manner to that suggested for the RMF transform in [11]. A comparison of results for the RMF and L across all 256 one variable functions of GF(2^2) is contained in the table below. Each set being optimised over 24 polarities. The number of functions realizable with *t* products is shown by:

t	RMF	L^{-1}
0	1	1
1	57	33
2	138	138
3	60	84
4	0	0
	2.004	2.191 (Expected Frequencies)

Our use of the basis represented by L is a special case of logic developed in [10]. Where it was shown that for n variables any set of 2^n functions that are linearly independent, when added using exclusive-or, will form a basis. Thus a transform in terms of these functions exists. As the only constraints on our choice of functions is linear independence we are free to change this basis into many alternative forms.

Considering the matrix L, as each X_i is Boolean, we can rewrite this transform in any one of four polarities. The last column being an **AND** function. We went further and choose as last columns all the Boolean functions that were linearly independent of the first three columns. These turned out to be the functions with either one logic 1 or three logic 1's and easy to implement. This gave a total of 32 possibilities. These transform functions can be used in any situation whether a state assignment technique is used or not. A software simulation was used to implement all 256 single variable functions. We were able to choose the polarity and logic function to minimise the number of coefficients these coefficients had an expected frequency of 2.051.

6. Conclusion

Field-Programmable Gate Arrays offer a promising structure for building the circuits suggested in this paper. Over GF(2^3) the homogeneous nature of the power of 'N' module in each coordinate means that the complete design is just the duplication of smaller units. The strapping arrangement for the constant times X module could be done in switches. In [12] Vranesic points to a possible future where the data running through a FPGA will be multi-valued but the control bits would remain digital. Using state assignments may bring us half way to this future. Using our GF(2^3) assignment a Universal Logic Module can be built that can implement any function. Inspecting Table 7 again we see that only three power of 'N' modules are actually needed. Other powers are got by permutation of inputs or simple logic. If constant times X modules are used to multiply by coefficients then we can consider the strapping arrangements as the control bits which generate any given function. Vranesic in [12] raised the coming possibility of implementing 4-valued Look Up Tables. This means we could see non-Boolean state assignments that can easily handle larger fields. For example GF(2^6) on three wires. Logic modules for 4EXP and 4AND can be designed using state assignment techniques. The 4EXP is similar to a X to the power of N module. Work still needs to be done to decide which assignment is optimal also to achieve RMF design in Larger fields. On the negative side state assignment techniques bring with them the disadvantages of extra wiring. At what stage this will cease to be economic as assignments are applied to larger fields remains to be found out.

References

[1] Menger, K.S., 'A transform for logic networks', IEEE Trans. Computers, Vol.C-18, No3, 1969, 241-251.

[2] Wang, S.J. and Horng, M.D.:'State assignment of finite state machines for low power applications' Electronics Letters 5th Dec 1996 Vol. 32 No. 25

[3] Falkowski, B.J and Rahardja,S.:'Generalised hybrid arithmetic canonical expansions for completely specified quaternary functions', IEE Proc-Circuits Syst., Vol 144 No 4 August 1997, pp. 201-208

[4] Stankovic, R.S., "Some remarks on Fourier transforms and differential operators for digital functions", *Proc. 22nd Int. Symp. on Multiple-Valued Logic,* May 27-29, 1992, Sendai, Japan, 365-370

[5] Stankovic, R.S., Moraga, C.: 'Reed-Muller-Fourier representations of multiple-valued functions over Galois fields of prime cardinalty", in: Kebschull, U., Schubert, E., Rosensteil, W., Eds, *Proc. IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design,* 16.-17.9.1993, Hamburg, Germany, 115-124

[6] Zilic, Z, Vranesic, Z.: 'Current CMOS Galois field circuits', *Proc. 23rd Int'l Symp. Multiple-Valued Logic,* pp. 245-250, May 1993.

[7] Godement, R.: 'Algebra' pp. 387, (Kershaw London 1969)

[8] Ireland, K. and Rosen, M: 'A Classical Introduction to Modern Number Theory', pp. 33 (Springer, 1992)

[9] Calingairt, P.: 'Switching function canonical forms based on commutative and associative binary operations', Trans. Amer.Inst. Elect. Eng., Vol. 79 Jan, 808-814 (1961)

[10] Perkowski, M.A., 'A fundamental theorem for EXOR circuits', *Proc. RM'93,* 1993, Hamburg, Germany, 52-60

[11] Stankovic, R., Jankovic, D. and Moraga, C.: 'Reed-Muller-Fourier verses Galois Field Representation of Four Valued Functions', Proceedings of the 28th IEEE International Symposium on Multi-Valued Logic, Fukuoka Japan, May 1998, pp. 186-191

[12] Vranesic, Z.G.:'The FPGA Challenge', Proceedings of the 29th IEEE International Symposium on multiple-valued Logic, Fukuoka Japan, May 1998, pp. 121-126

Table 5. A two variable function over GF(2^2)

z_1		z_2		$F(Z_1,Z_2)$
x_4	x_3	x_2	x_1	
0	0	0	0	0
0	0	0	1	3
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	3
0	1	1	0	2
0	1	1	1	3
1	0	0	0	2
1	0	0	1	2
1	0	1	0	3
1	0	1	1	2
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	2

Table 6. Constant times X

	Y Times X Co-ordinates		
Y	First	Second	Third
e_0	0	0	0
e_1	x_1	x_2	x_3
e_2	x_1+x_3	x_3	$x_1+x_2+x_3$
e_3	x_3	x_2+x_3	x_1+x_2
e_4	x_2	$x_1+x_2+x_3$	x_2+x_3
e_5	x_1+x_2	x_1+x_3	x_2
e_6	$x_1+x_2+x_3$	x_1+x_2	x_1
e_7	x_2+x_3	x_1	x_1+x_3

Table 3. An assignment for $GF(2^3)$

ELEMENT	ASSIGNMENT		
x	x_1	x_1	x_2
e_0	0	0	0
e_1	1	1	1
e_2	0	1	1
e_3	1	0	0
e_4	1	1	0
e_5	0	0	1
e_6	1	0	1
e_7	0	1	0

Table 4. Menger's assignment

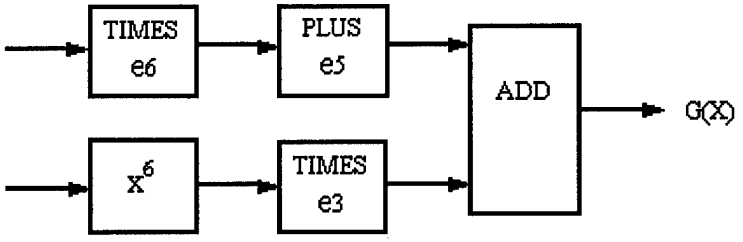
ELEMENT	ASSIGNMENT		
x	x_1	x_2	x_3
e_0	0	0	0
e_1	0	1	0
e_2	0	0	1
e_3	0	1	1
e_4	1	0	0
e_5	1	1	0
e_6	1	0	1
e_7	1	1	1

Table 7. X^n

Assignment Co-ordinates

	First	Second	Third
X	x_1	x_2	x_3
X^2	x_2	x_3	x_1
X^3	$X_3+x_1+(X_1X_2)$	$X_1+x_2+(X_2X_3)$	$X_2+x_3+(X_3X_1)$
X^4	x_3	x_1	x_2
X^5	$X_2+x_3+(X_3X_1)$	$X_3+x_1+(X_1X_2)$	$X_1+x_2+(X_2X_3)$
X^6	$X_1+x_2+(X_2X_3)$	$X_2+x_3+(X_3X_1)$	$X_3+x_1+(X_1X_2)$
X^7	x_1 0R x_2 0R x_3	x_1 0R x_2 0R x_3	x_1 0R x_2 0R x_3

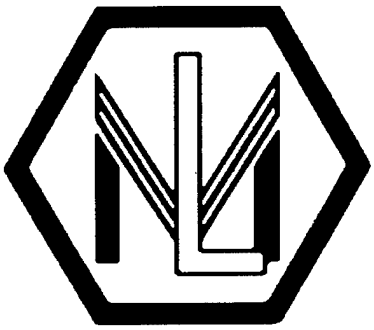
Figure 1. A network for $G(X)= e_5+ e_6+ e_3x_6$



SESSION VIIB

LOGIC

CHAIR: Lucien Haddad



Information Relationships and Measures in Application to Logic Design

Lech Jóźwiak
Eindhoven University of Technology
Faculty of Electrical Engineering
P.O. Box 513, EH 10.25
5600 MB Eindhoven, The Netherlands
e-mail:LECH@ics.ele.tue.nl

Abstract

In this paper, the theory of information relationships and relationship measures is considered and its application to logic design is discussed. This theory makes operational the famous theory of partitions and set systems of Hartmanis. The information relationships and measures enable us to analyze relationships between the modeled information streams and constitute an important analysis apparatus that can be used for analysis and synthesis of various information systems. It can be applied in logic design, pattern analysis, machine learning, and other fields. This paper shows how to apply the relationships and measures to logic design when using as examples three important problems: input support minimization, parallel decomposition and serial functional decomposition. The application examples and experimental results presented in the paper demonstrate the high potential of the information relationships and measures in solving logic synthesis problems.

1. Introduction

The opportunities created by modern microelectronic technology cannot be fully exploited, because of weaknesses of the traditional logic synthesis methods. Particularly in the case of (C)PLDs, look-up table FPGAs and complex CMOS gates, the constraints are imposed not on the function type a certain block can implement, but on various building block structural parameters (e.g. the number of inputs, outputs, product terms in a building block or the number of serial and parallel transistors in a gate) and on interconnections between the building blocks. A block is able to implement any function with limited dimensions. On the other hand, the traditional logic synthesis methods do not consider hard structural constraints. In principle, they are devoted to only some very special cases of possible implementation structures involving some minimal functionally complete systems of logic gates (e.g. AND+OR+NOT, AND+EXOR, MUX). They require a post synthesis technology mapping for another implementation structures. If the actual synthesis target strongly differs from these minimal systems, e.g. involves a lot of complex gates, look-up table FPGAs or (C)PLDs, any technology mapping cannot guarantee a good result if the initial synthesis was performed without close relation to the actual target.

Therefore, there is presently much research in the field of general (functional) decomposition of combinational circuits and sequential machines [3]-[10] [12] [14] [15] [17] [18] [20]-[22]. The most promising recent approaches in pattern analysis, knowledge discovery, machine learning, decision systems, data bases, data mining etc. are also based on general functional decomposition [11] [12] [13] [16] [19] [23]. In [5], I presented the theory of general decomposition and explained how the theory can be used for the synthesis of sequential and combinational circuits. For large circuits however, the number of possible decompositions is so great that it becomes necessary to construct only the most promising decompositions, using the general decomposition theorems presented in [5] together with the appropriate heuristic evaluation functions and selection mechanisms. These evaluation and selection mechanisms must limit the search space to a manageable size while keeping high-quality solutions in the limited space. In particular, the analysis and evaluation of relationships between information in various information streams of a considered system is here of primary importance. In [8] I proposed the fundamental analysis apparatus for this aim: the theory of information relationships and measures.

The information relationships and measures serve for analysis and estimation of information and information interrelationships in discrete systems, as modeled by any sort of discrete relations, functions and sequential machines. They can be applied for both the binary as well as the multiple-valued and symbolic systems, in many fields of modern engineering and science, including logic and architecture synthesis for VLSI systems [3] [10] [12] [14] [15] [17] [18] [20]-[22], pattern analysis, knowledge discovery, machine learning, decision systems, data bases, etc. [11][12][13][16][19][23]. Results of the relationship

0-7695-0161-3/99 $10.00 © 1999 IEEE

analysis make it possible to discover the nature of the considered system or to decide its structure in order to satisfy given design constraints or optimize certain objectives.

The theory of information relationships and measures makes operational the famous theory of partitions and set systems of Hartmanis [2]. Partitions and set systems enable us to model information streams. The information relationships and measures enable us to analyze the relationships between the modeled information streams.

2. Representation of information in discrete information systems

Information is represented in discrete systems by values of some signals or variables. Let's consider a certain finite set of elements S called symbols. Information about symbols (elements of S) means the ability to distinguish certain symbols from some other symbols. Knowing a certain value of a certain attribute, signal or variable x, it is possible to distinguish a certain subset B of elements from S from all other elements of S, but it is impossible to distinguish between the elements from B. For example, if we consider an incompletely specified multiple-output Boolean function in Fig. 1, where symbols 1,...,5 represent terms (cubes) on input variables x_1,..., x_6, then different values of the input variable x_3 enable us to distinguish between the subset {1, 2, 3} and {4, 5}. For 1, 2 and 3: $x_3 = 0$ and for 4 and 5: $x_3 = 1$. Thus, knowing that $x_3 = 0$, we know that 1, 2 or 3 occurred, but we do not know which of the three symbols occurred. Knowing that $x_3 = 1$, we know that 4 or 5 occurred, but we do not know which of the two symbols occurred. In such a way information is modeled with partitions and set systems [1][2][5].

A **set system** SS on a set S is defined as a collection of nonempty and distinct subsets $B_1, B_2, \ldots, B_k$ of S called blocks, such that:

$\bigcup_i B_i = S$ and $B_i \not\subset B_j$ for $i \neq j$

A **generalized set system** (blanket [23]) SS on a set S is defined as a collection of nonempty and distinct subsets $B_1, B_2, \ldots, B_k$ of S such that: $\bigcup_i B_i = S$.

The set system algebra is presented in [1]. The **product and sum operations** are defined **for set systems** as follows:

$SS_1 \bullet SS_2 = \mathrm{Max}\,(\,\{B_{SS1} \cap B_{SS2} \mid B_{SS1} \in SS_1 \wedge B_{SS2} \in SS_2\}\,)$, and $SS_1 + SS_2 = \mathrm{Max}\,(\,SS_1 \cup SS_2\,)$, where: $\mathrm{Max}(\,\{B_i \mid B_i \subseteq S\}\,) = \{B \mid B \in \{B_i\} \wedge ((B' \in \{B_i\} \wedge B \subseteq B') \Rightarrow B = B')\}$ (Max removes blocks contained in other blocks).

SS_1 is **smaller than or equal** to SS_2: $SS_1 \leq SS_2$ if and only if each block of SS_1 is included in a block of SS_2.

A set system SS on S can be interpreted as a compatibility relation defined on S, with the compatibility classes being the blocks of SS. Such a compatibility relation is reflexive and symmetric but is not required to be transitive. If it is transitive, i.e. the subsets B_i of the corresponding set system SS are disjoined, then it is an equivalence relation and the set system SS is a **partition**. A certain set system SS (partition P) gives information about the elements of S with limited precision to the compatibility (equivalence) class. With this information it is possible to distinguish the different classes although it is impossible to distinguish between the elements of the same class. The set system product can be interpreted as a product of the corresponding relations; it represents combined information about the elements of S that is provided by the relations together. The set system sum can be interpreted as the sum of the corresponding relations and it represents information about elements of S after applying the combined abstraction provided by the relations involved. The partial ordering relation denotes the fact that if $SS_1 \leq SS_2$, then SS_1 provides the same or more information about elements of S than SS_2. In this paper, we will use the term *set system* to designate a set system, generalized set system or partition, everywhere where this will not lead to misunderstanding.

term symbols	inputs x_1	x_2	x_3	x_4	x_5	x_6	outputs y_1	y_2	y_3	y_4
1	0	-	0	0	0	1	0	1	1	0
2	0	0	0	1	0	-	1	1	-	0
3	1	-	0	0	1	-	1	1	1	0
4	1	1	1	1	1	-	0	-	0	1
5	-	0	1	-	0	0	-	0	-	0

Figure 1. Incompletely specified multi-output Boolean function

When analyzing or designing information systems, we are interested in relationships between information streams which express such properties as similarity (common information), dissimilarity (different information), missing information, extra information, etc. If we want to consider the interesting relationships and to base the design decisions on results of such analysis, introduction of an adequate analysis apparatus is necessary. This is however impossible before answering the question: **"what information is modeled by a certain set system?"** To answer this question, it was necessary to discover what is an elementary (atomic) portion of information, i.e. such portion that any other information (for example as modeled by a set system) can be expressed as a composition of such atomic portions.

I proposed the following solution [8]: an **elementary information** describes the ability to distinguish a certain single symbol s_i from another single symbol s_j, where: s_i, $s_j \in S$ and $s_i \neq s_j$. Any set of atomic portions of information can be represented by an **information relation I, information set IS** and **information graph IG** defined on $S \times S$ as follows:

$I = \{(s_i, s_j) \mid s_i$ is distinguished from s_j by the modeled information$\}$,

IS = {{s_i, s_j} | s_i is distinguished from s_j by the modeled information}, and

IG = {S, {{s_i, s_j} | s_i is distinguished from s_j by the modeled information}}.

In a strictly analogous way, the notion of the elementary (atomic) abstraction, and the appropriate abstraction (compatibility) relation A, set AS and graph AG can be introduced [8].

Relationships between set systems can now be analyzed by considering the relationships between their corresponding information and abstraction relations, sets and graphs. In particular, the correspondence between SS, IS and AS is as follows: IS contains the pairs of symbols that are not contained in any single block of a corresponding SS, and AS contains the pairs of symbols that are contained in at least one single block of a corresponding SS.

Example 1. (information modeling with set systems and information sets)

For the function from Fig. 1 the appropriate set systems and information sets for particular input and output variables and for the whole 4-output function are listed below:

$SS(x_1)$={{1,2,5};{3,4,5}} $IS(x_1)$={1|3, 1|4, 2|3, 2|4},
$SS(x_2)$={{1,2,3,5};{1,3,4}} $IS(x_2)$={1|4; 2|4; 4|5},
$SS(x_3)$={{1,2,3};{4,5}} $IS(x_3)$={1|4,1|5,2|4,2|5,3|4,3|5},
$SS(x_4)$={{1,3,5};{2,4,5}} $IS(x_4)$={1|2, 1|4, 2|3, 3|4},
$SS(x_5)$={{1,2,5};{3,4}} $IS(x_5)$={1|3,1|4,2|3,2|4,3|5,4|5},
$SS(x_6)$={{1,2,3,4};{2,3,4,5}} $IS(x_6)$={1|5}.

$SS(y_1)$={{1,4,5};{2,3,5}} $IS(y_1)$={1|2, 1|3, 2|4, 3|4},
$SS(y_2)$={{1,2,3,4};{4,5}} $IS(y_2)$={1|5; 2|5; 3|5},
$SS(y_3)$={{1,2,3,5};{2,4,5}} $IS(y_3)$={1|4,3|4},
$SS(y_4)$={{1,2,3,5};{4}} $IS(y_4)$={1|4, 2|4, 3|4, 4|5}.

$SS(y_1,y_2,y_3,y_4)$={{1};{2,3};{4};{5}}
$IS(y_1,y_2,y_3,y_4)$={1|2,1|3,1|4,1|5,2|4,2|5,3|4,4|5,3|5,4|5} ∎

To stress that the symbols from a pair of incompatible symbols are distinguished from each other and to shorten the notation of information sets, we will further denote each pair of incompatible symbols {s_i, s_j} by $s_i|s_j$.

In the same way information can be modeled with set systems and information sets for any discrete system, including multiple-valued systems. For instance in the case of a multiple-valued function, the only difference compared to the binary function is the fact, that the set systems induced by particular input and output variables are the multiple-block set systems instead of two-block set systems. Each of them has as many blocks as many values can take a certain input/output variable.

3. Information relationships and measures

When performing analysis or design of information systems, we often ask for relationships between information in various information streams, places or parts of a considered system. In general, we are interested in information relationships that express similarity, dissimilarity, missing information, extra information etc. Below, some basic information relationships are recalled which express these elementary properties:

- **common information** CI (i.e. information that is present in both SS_1 and SS_2): $CI(SS_1, SS_2) = IS(SS_1) \cap IS(SS_2)$
- **total** (combined) **information** TI (i.e. information that is present either in SS_1 or in SS_2): $TI(SS_1, SS_2) = IS(SS_1) \cup IS(SS_2)$
- **missing information** MI (i.e. information that is present in SS_1, but missing in SS_2): $MI(SS_1, SS_2) = IS(SS_1) - IS(SS_2)$
- **extra information** EI (i.e. information that is missing in SS_1, but present in SS_2): $EI(SS_1, SS_2) = IS(SS_2) - IS(SS_1)$
- **different information** DI (i.e. information that is present in one of the set systems and missing in the other): $DI(SS_1, SS_2) = MI(SS_1, SS_2) \cup EI(SS_1, SS_2)$.

Analogous relationships can be defined for abstraction [8].

Example 2. (some information relationships for the function from Fig. 1)

$CI(SS(y_1), SS(x_4))$ = {1|2, 3|4}
$TI(SS(y_1), SS(x_4))$ = {1|2, 1|3, 1|4, 2|3, 2|4, 3|4}
$MI(SS(y_1), SS(x_4))$ = {1|3, 2|4}
$EI(SS(y_1), SS(x_4))$ = {1|4, 2|3}
$DI(SS(y_1), SS(x_4))$ = {1|3, 1|4, 2|3, 2|4} ∎

With the relationship apparatus defined above, we can answer such questions as: what information required to compute values of a certain variable is present in another variable, what information is missing, what extra information is present etc.

However, knowledge of "what" is often not sufficient to take appropriate design decisions, because the design decisions are based in most cases not only on some qualitative, but also on some quantitative comparisons. Along the knowledge of "what", some knowledge of "how much" and "how important" is required. For instance: how much of the required information provides a certain variable and how much another one, is it much more, etc. Moreover, information that is provided by just a single input may be considered as more important than information provided by all inputs, an input that delivers a unique information may be considered as more important than another input that delivers only information provided also by some other inputs etc.

Therefore, some quantitative relationship measures are often necessary.

For a single set system *SS* the **information quantity IQ** is defined: IQ(SS) = |IS(SS)|.

For two set systems SS1 and SS2, the following **relationship measures** are defined:

- **information similarity (affinity) measure ISIM**: $ISIM(SS_1, SS_2) = |CI(SS_1, SS_2)|$
- **information dissimilarity (difference) measure IDIS**: $IDIS(SS_1, SS_2) = |DI(SS_1, SS_2)|$
- **information decrease (loss) measure IDEC**: $IDEC(SS_1, SS_2) = |MI(SS_1, SS_2)|$
- **information increase (growth) measure IINC**: $IINC(SS_1, SS_2) = |EI(SS_1, SS_2)|$
- **total information quantity TIQ**: $TIQ(SS_1, SS_2) = |TI(SS_1, SS_2)|$.

It is also possible to define some relative measures, by normalizing the above absolute measures, and weighted measure, for example, by associating an appropriate importance weight $w(s_i|s_j)$ with each elementary information [8]. In a strictly analogous way, the abstraction relationship measures can be defined [8].

4. Application of the relationships and measures to logic synthesis

The information and abstraction relationships as well as the measures for the strength and importance of the relationships enable designers and tools to analyze relationships between the information streams of the considered systems and provide them with data necessary for effective and efficient decision-making. Results of the relationship analysis make it possible to discover the nature of the considered system or to decide its structure in order to satisfy given design constraints or optimize certain objectives.

In order to demonstrate the importance of the information relationship analysis and to show how to apply it in the logic synthesis process, let's consider the problems of the input support minimization, parallel (output) decomposition and serial functional decomposition. These problems are highly relevant and common for both the binary logic applications (e.g. VLSI circuit design [3]-[10][12][14][15][17][18][20]-[22]) and multiple-valued logic applications (e.g. pattern analysis, knowledge discovery, machine learning, decision systems, data bases, VLSI circuit design etc. [11][12][13][16][19][23]). They are considered below for the case of incompletely specified Boolean functions. However application of the information relationships and measures to the multiple-valued logic is strictly analogous, because in the multiple-valued case the information is modeled with information sets the same way as in the binary case, and the relationships and measures operate on the information sets.

4.1. Application to input support minimization.

Input support minimization consists of finding a minimal sub-set of inputs that contains all information that was required to compute by the originally specified function, or sequential machine [7][9].

Let's find the minimum support for the function from Fig. 1, by analyzing the relationships between the information required for computing the output values (modeled by set system $SS(y_1,y_2,y_3,y_4)$= {{1};{2,3};{4};{5}}) and information provided by particular input variables. Some of the interesting relationships and the corresponding measures are given below:

$CI(y, x_1) = \{1\|3, 1\|4, 2\|4\}$	$\|CI(y, x_1)\| = 3$
$CI(y, x_2) = \{1\|4, 2\|4, 4\|5\}$	$\|CI(y, x_2)\| = 3$
$CI(y, x_3) = \{1\|4, 1\|5, 2\|4, 2\|5, 3\|4, 3\|5\}$	$\|CI(y, x_3)\| = 6$
$CI(y, x_4) = \{1\|2, 1\|4, 3\|4\}$	$\|CI(y, x_4)\| = 3$
$CI(y, x_5) = \{1\|3, 1\|4, 2\|4, 3\|5, 4\|5\}$	$\|CI(y, x_5)\| = 5$
$CI(y, x_6) = \{1\|5\}$	$\|CI(y, x_6)\| = 1$

Observe that:

$|CI(y, x_3)| > |CI(y, x_5)| > |CI(y, x_1)| = |CI(y, x_2)| = |CI(y, x_4)| > |CI(y, x_6)|$.

The information similarity measure tells us, that x_3 delivers more information that is required for computing y than x_5, x_5 more than x_1, etc. Furthermore, x_3 delivers a unique information that is necessary for computing the output, but is not delivered by any other input variable, namely: 2|5. Therefore, x_3 must be in any support, thus also in any minimal support. Since $CI(y, x_3)$ does not cover IS(y) completely, the symbols from the pairs given by $IS(y) - CI(y, x_3) = \{1|2, 1|3, 4|5\}$ are not distinguished from each other by x_3, and therefore at least one extra input variable is necessary for the minimum support. Because $CI((IS(y) - CI(y, x_3)), x_1) = \varnothing$, $CI((IS(y) - CI(y, x_3)), x_2) = \{4|5\}$, $CI((IS(y) - CI(y, x_3)), x_4) = \{1|2\}$, $CI((IS(y) - CI(y, x_3)), x_5) = \{1|3, 4|5\}$, $CI(y, x_6) = \varnothing$, no single variable delivers the lacking information, but x_4 and x_5 together provide it. This means that x_3, x_4 and x_5 constitute the minimum input support.

In a similar way the information relationships and weighted measures are used in our genetic algorithm (GA) for finding the minimal input support [7]. In particular, the weighted information similarity measure is used in GA for the semi-random construction of the initial support population. The probability of a certain input variable to be selected for an initial support is proportional to the value of this measure. The same measure is also used for the selection of input variables in the local search performed by the deterministic *merge* and *repair* operators of the genetic algorithm [7]. Table 1 presents the test results of GA compared to the strictly minimal results and results from *QuickScan* algorithm [7][9] computed at the HP9000/735 processor under HP-UX 9.05. The results from GA are computed quickly and are in almost all cases strictly optimal and sometimes on distance one from optimum. The test files were specially generated, because the standard logic synthesis benchmark set contained functions with minimal input supports. The benchmarks do not contain either essential or dominated inputs and are "difficult", because they cannot be solved or reduced by the preprocessing

Table 1. Test results of the genetic algorithm (GA) and *QuickScan* (QS).

File	Q (min)	Q QS	T QS	Q GA-[1]	T GA-[1]	Q mg	T mg	Q GA[2]	T GA[2]
i10	7	7	<1	7	<1	7	<1	7	<1
i20s04a	4	4	<1	4	1	4	<1	4	<1
i20s05a	5	5	<1	5	1	5	<1	5	2
i20s06a	6	6	<1	6.17	2	6	<1	6	2
kaz	5	6	<1	5.17	2	5	<1	5	1
i30s08a	8	9	2	9.17	30	8	6	8	40
i30s08b	8	9	2	9.17	22	9	6	8	38
i30s08c	8	9	2	9.17	21	8	6	8.13	34
i30s10a	10	11	6	11.17	1:37	11	58	10.88	2:39
i40s09a	9	10	5	9.83	56	9	19	9	1:52
i40s09b	9	10	5	9.83	55	9	19	9	1:53
i40s10a	10	11	9	11.17	2:53	11	1:47	10	6:47
i40	10	11	9	11	3:09	11	1:47	10	6:32
i50	7	9	2	8.67	20	8	4	8	48
i100	9	11	30	10.83	3:31	9	1:02	9	9:58

Q-quality (number of variables in the support) of the best support found, Q(min) – quality of the minimum support, T – computation time (min:sec), QS – QuickScan, GA – genetic algorithm, GA- - GA without merge, mg – merge performed on the full original support, [1] Average over 6 runs, [2] Average over 8 runs.

techniques (i.e. recursively removing the rows covered by essential columns, and the dominated columns and rows). The number by "i" in the benchmark's name is equal to the number of input variables of the benchmark. More complete characteristics of the benchmarks and of the way they were generated as well as benchmark results of another input support minimization programs can be found in [7] and [9].

4.2. Application to parallel (output) decomposition.

Parallel decomposition is a very important decomposition technique used for multiple output functions, relations and sequential machines [4][6][12][17]. In parallel decomposition, computations of values for certain subsets of outputs are performed in separate blocks. Thus, parallel decomposition consists of partitioning of the set of output variables in disjoint subsets, each satisfying the building block constraints, so that some objectives are optimized and some other possible constraints satisfied.

Assume that each logic building block for implementing the considered example function can implement any function, with maximally 2 inputs and 2 outputs (as in look-up table FPGAs). Since the example function has 4 outputs and requires minimum 3 inputs, it is impossible to implement the function with a single building block. The function must be decomposed into at least two blocks. Find a parallel decomposition with minimal number of blocks and minimum interconnections.

Let's compute the affinity relationships between the information required by particular output variables and information delivered by particular input variables and the corresponding relative similarity measures. For simplicity sake, let's consider only the minimal input support $\{x_3, x_4, x_5\}$ (the function with the original support can be considered precisely the same way).

$CI(y_1, x_3) = \{2|4, 3|4\}$ $ISIM_r(y_1, x_3) = 0,5$
$CI(y_1, x_4) = \{1|2, 3|4\}$ $ISIM_r(y_1, x_4) = 0,5$
$CI(y_1, x_5) = \{1|3, 2|4\}$ $ISIM_r(y_1, x_5) = 0,5$
$CI(y_2, x_3) = \{1|5, 2|5, 3|5\} = IS(y_2)$ $ISIM_r(y_2, x_3) = 1$
$CI(y_2, x_4) = \varnothing$ $ISIM_r(y_2, x_4) = 0$
$CI(y_2, x_5) = \varnothing$ $ISIM_r(y_2, x_5) = 0$
$CI(y_3, x_3) = \{1|4, 3|4\} = IS(y_3)$ $ISIM_r(y_3, x_3) = 1$
$CI(y_3, x_4) = \{1|4, 3|4\} = IS(y_3)$ $ISIM_r(y_3, x_4) = 1$
$CI(y_3, x_5) = \{1|4\}$ $ISIM_r(y_3, x_5) = 0,5$
$CI(y_4, x_3) = \{1|4, 2|4, 3|4\}$ $ISIM_r(y_4, x_3) = 0,75$
$CI(y_4, x_4) = \{1|4, 3|4\}$ $ISIM_r(y_4, x_4) = 0,5$
$CI(y_4, x_5) = \{1|4, 2|4, 4|5\}$ $ISIM_r(y_4, x_5) = 0,75$

Since $ISIM_r(y_2, x_3) = 1$ and $ISIM_r(y_3, x_3) = 1$, x_3 delivers complete information required for computing y_2 and y_3. Thus, y_2 and y_3 can be implemented with just a single logic block with one input x_3 and two outputs y_2 and y_3. y_1 and y_4 require each at least two input variables for their computation, because no single variable delivers all required information ($ISIM_r(y_1, x_i)<1$ and $ISIM_r(y_4, x_i)<1$, for all i=3,4,5). Furthermore, x_5 provides a unique information for y_1 (1|3), and for y_4 (4|5), not provided by any other input. Importance of x_5 is thus highest, and therefore, x_5 must be used both for computing y_1 and y_4. Similarly, x_4 provides a unique information for y_1 (1|2), importance of x_4 is also highest and x_4 must be used for computing y_1. Since $CI(y_1, \{x_4, x_5\}) = \{1|2, 1|3, 2|4, 3|4\} = IS(y_1)$, $\{x_4, x_5\}$ is a complete minimum support for y_1. $\{x_4, x_5\}$ is also a complete minimal support for y_4, because:

$CI(y_4, \{x_4, x_5\}) = \{1|4, 2|4, 3|4, 4|5\} = IS(y_4)$.

Thus, y_1 and y_4 can be implemented with just a single building block with two inputs x_4 and x_5, and two outputs y_1 and y_4. In this way, we constructed a parallel decomposition of the considered function (Fig. 2) that satisfies the input and output constraints of the used building blocks, uses minimum number of building blocks (just two blocks) and minimum interconnections (no common inputs to different blocks).

More information on parallel decomposition, prototype tools we developed for this aim and some benchmark results of these tools can be found in [4] and [6].

4.3. Application to serial functional decomposition.

Fig. 3 shows the schematic representation of the serial functional decomposition.

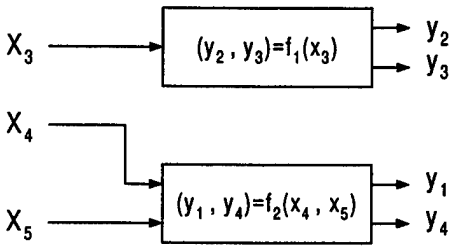


Figure 2. Optimal parallel decomposition of the example function from Fig. 1.

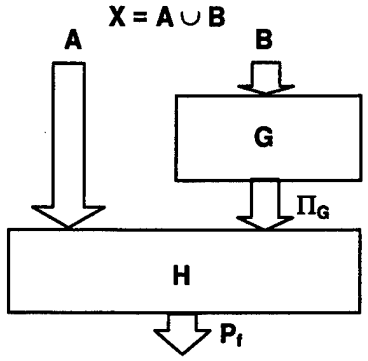


Figure 3. Serial decomposition scheme

Let X be the set of input variables of a given function F. Let A and B be two subsets of X such that $A \cup B = X$. Let P(A) be a set system induced by the input variables from the set A on the set of the function terms (cubes) F, let P(B) be a set system induced on F by the variables from B and P_F be a set system induced on F by the output variables of F.

Theorem 1 If there exists a set system Π_G on F such that $P(B) \leq \Pi_G$, and $P(A) \bullet \Pi_G \leq P_F$, then F has a serial decomposition with respect to (A,B) [17]. ∎

The conditions for serial decomposition, can be re-expressed in terms of the information sets in the following way:

Theorem 2. If there exists a set system Π_G on F such that $IS(P(B)) \supseteq IS(\Pi_G)$, and $IS(P(A) \bullet \Pi_G) \supseteq IS(P_F)$, then F has a serial decomposition with respect to (A, B) [20]. ∎

In general, information that is necessary for computing values of a certain output of a considered multiple-output function is distributed on a number of its inputs. The inputs also deliver some information, that is not needed for the output, and information on the inputs is represented otherwise than on the considered output. Block G can be therefore considered as a block where an intermediate information transformation is performed which involves an appropriate combination, abstraction and re-structuring of the input information. This transformation consists of construction of an appropriate set system Π_G from a selected set system P(B). The set system Π_G should carry a part of the information delivered by the set system P(B), which in combination with information delivered by the set system P(A), is essential to compute the required output information (Theorem 2). Since the number of block's G outputs should be in practical cases much smaller than the number of its inputs, Π_G must have much less blocks than P(B). Π_G is created by merging the blocks of P(B) and a lot of abstraction is made during this merging. To avoid too big loss of information during the merging process or creation of set systems Π_G with too large number of blocks (which would require too many physical logic blocks in implementation), set B should contain input variables that carry relatively much information which is not important for computing values of the output variables. Moreover, a part of the important information delivered by variables from set B is in most cases also delivered by some transfer this information to the output of G. It can be abstracted during the merging process. Also, the set systems induced by each of the input variables from set B should be similar each to another, because this results in less blocks of P(B) itself and less merging to perform.

Based on these observations the following rules of input variable selection for set B have been developed. Variable x_i should be included into set B if:

IINC(P_F, P(x_i)) is relatively high,

ISIM(P(A), P(x_i)) is relatively high,

ISIM(P(B'), P(x_i)) is relatively high,

where: x_i - candidate to set of indirect variables B, B' – partially created set B.

Using these rules, we developed and implemented a heuristic method for the (near-) optimal input support selection in serial decomposition. Experimental results from our method are presented below and compared to the results from the systematic search. In the systematic search the supports are implicitly enumerated. For each support, the corresponding set system Π_G is computed and the first found support with the minimum number of blocks in Π_G is used for decomposition. The number of blocks of partition Π_G shows strong positive correlation with the number of logic blocks and logic levels in the resulting decomposition, and therefore, it is a good measure for the support quality. In the case of our heuristic selection also the first support is used for decomposition, but the first one found based on information relationships measures. We used for experiments a number of functions from the international logic synthesis benchmark set. Table 2 and 3 report results of the input support selection for set B in a single serial decomposition step, as illustrated in Fig. 2. Table 4 shows the results of decomposition of some benchmark functions into a network of 4-input, 1-output logic cells, obtained by repeating the serial decomposition step from Fig. 3 a number of times. In Table 2 the comparison of the number of blocks of partition Π_G is presented for heuristic and systematic method.

Table 2. Comparison of the number of blocks in partition Π_G obtained by the systematic and heuristic method.

Bench-mark	Size			Systematic method (|B|)				Heuristic method (|B|)			
	in-puts	out-puts	cubes	3	4	5	6	3	4	5	6
z4	7	4	128	4	6	8	12	4	6	8	12
5xp1	7	10	126	8	16	32	64	8	16	32	64
misex1	8	7	18	4	6	7	9	4	6	7	9
root	8	5	71	5	9	15	17	5	9	15	17
opus	9	10	23	4	6	8	10	5	6	8	10
9sym	9	1	191	4	5	6	7	4	5	6	7
alu2	10	3	391	6	12	24	43	6	12	24	43
sao2	10	4	60	4	6	9	11	4	6	9	11
sse	11	11	39	4	6	8	11	4	6	9	11
keyb	12	7	147	6	9	13	19	6	10	14	19
s1	13	11	110	5	8	13	19	6	10	15	19
plan	13	25	115	5	7	11	17	5	9	14	19
styr	14	15	140	4	6	9	13	5	7	10	13
ex1	14	24	127	4	6	8	11	4	6	8	11
kirk	16	10	304	4	4	5	6	4	5	5	6

The systematic search always finds a support, which results in partition Π_G with the strictly minimum number of blocks. However, the method based on information measures produces also the minimal or near-minimal results and does it much faster. Table 3 shows comparison of the computation time for both methods for a single support selection step. In Table 4, the number of logic cells (4-input, 1-output FPGA cells) in decompositions is presented for several benchmarks, for decompositions found by application of the heuristic or systematic support selection a number of times. The method based on information relationship measures produces decompositions with the same or almost the same number of blocks as the systematic method. In some cases the results from the heuristic support selection are even better than from the systematic. More information on the application of the information relationships and measures to the serial functional decomposition is presented in [18].

5. Conclusion.

The famous theory of partitions and set systems of Hartmanis [2] and the theory of information relationships and measures discussed here form just together an information modeling and analysis apparatus that is of primary importance for analysis and synthesis of discrete information systems, and in particular for logic design.

We applied the apparatus of information relationships and measures for design decision making in a number of prototype CAD tools for solving various logic design problems. Among others, we applied this apparatus for solving some logic decomposition problems and finding the minimal input support. The experimental results obtained with our prototype tools are very encouraging. For example, our input support minimization tool is able

Table 3. Comparison of the computation time (in seconds) for the systematic and heuristic method.

Bench-mark	Size			Systematic method (|B|)				Heuristic method (|B|)			
	in puts	out puts	cubes	3	4	5	6	3	4	5	6
z4	7	4	128	1	1	2	2	1	2	5	8
5xp1	7	10	126	0	2	1	2	2	2	5	7
misex1	8	7	18	0	1	1	3	1	1	2	5
root	8	5	71	1	2	2	3	1	2	5	12
opus	9	10	23	1	2	3	8	0	1	3	7
9sym	9	1	191	10	20	41	79	5	8	16	34
alu2	10	3	391	33	51	64	63	32	40	60	98
sao2	10	4	60	3	6	14	36	1	2	6	21
sse	11	11	39	2	7	20	63	1	2	8	25
keyb	12	7	147	17	52	156	503	7	11	24	51
s1	13	11	110	13	46	137	552	7	10	17	33
plan	13	25	115	16	51	184	595	6	9	19	42
styr	14	15	140	31	109	404	1453	12	17	32	58
ex1	14	24	127	24	91	333	1377	8	12	26	58
kirk	16	10	304	108	528	2125	11234	55	69	119	230

Table 4. Comparison of the number of logic cells in decomposition.

	Size			Systematic method	Heuristic Method
Benchmark	inputs	outputs	cubes		
z4	7	4	128	7	7
5xp1	7	10	126	20	21
misex1	8	7	18	19	18
root	8	5	71	46	47
alu2	10	3	391	116	114

to compute the minimal supports for difficult and large (100 inputs) problem instances within few minutes. Our prototype multi-level decomposition tool for FPGAs is able to perform a single step of the functional decomposition more than 50 times faster than the tool based on systematic search, on benchmarks with 14-16 inputs, 10-20 outputs and 100-300 cubes, when producing the strictly optimal or near optimal results. By repetitively applying the decomposition step, our tool is able to completely decompose such a benchmark into 4-input CLBs thousands times faster then the systematic algorithm, when producing the minimum number of CLBs or at most 1 CLB more than the minimum on some of the checked benchmarks. The difference in processing time between these two tools grows very fast with the size of the decomposed function.

The application examples and experimental results from our prototype tools demonstrate the high potential of the information relationships and measures in solving various logic analysis and synthesis problems

6. Acknowledgements

The author is indebted to Prof. T. Luba for his cooperation in the part of the research concerning

the serial functional decomposition and to M. Rawski and N. Ederveen for their help in implementation of programs and performing experiments.

7. References

[1] J.A. Brzozowski and T. Luba: Decomposition of Boolean Functions Specified by Cubes, University of Waterloo Research Report, CS-97-01, Waterloo, Canada, January 1997.

[2] J. Hartmanis, R.E. Stearns: Algebraic Structure Theory of Sequential Machines, Englewood Cliffs, N.J.: Prentice-Hall, 1966.

[3] S.C. Chang, M. Marek-Sadowska, T.T. Hwang: Technology Mapping for TLU FPGAs Based on Decomposition of Binary Decision Diagrams, Ieee Tr. On CAD, vol. 15, No 10, Oct. 1996, pp.1226- 1236.

[4] L. Jóźwiak, F. Volf: An Efficient Method for Decomposition of Multiple Output Boolean Functions and Assigned Sequential Machines, EDAC - The European Conference on Design Automation, Brussels, Belgium, March 16-19, 1992, pp. 114-122.

[5] L. Jóźwiak: General Decomposition and Its Use in Digital Circuit Synthesis, VLSI Design, vol.3, No 3, pp. 225 - 248, 1995.

[6] L. Jóźwiak, F.A.M. Volf: Efficient Decomposition of Assigned Sequential Machines and Boolean Functions for PLD Implementations, IEEE International Conference on Electronic Technology Directions, Adelaide, Australia, 23-25 May, 1995, pp. 258-266.

[7] L. Jóźwiak, N.Ederveen: Genetic Algorithm for Input Support Minimization, International Conference on Computational Intelligence and Multimedia Applications (ICCIMA?98), Melbourne, Australia, Febr. 9-11, 1998.

[8] L. Jóźwiak: Information Relationships and Measures - An Analysis Apparatus for Efficient Information System Synthesis, Proceedings of the 23rd EUROMICRO Conference (EUROMICRO'97), Budapest, Hungary, September 1-4, 1997, pp. 13-23., IEEE Computer Society Press.

[9] P.A. Konieczny and L. Jóźwiak: Minimal Input Support Problem and Algorithms to Solve It, Eindhoven University of Technology Research Reports, EUT Report 95-E-289, Eindhoven University of Technology, The Netherlands, Apr. 1995.

[10] Y.T. Lai, K.R.R. Pan, M. Pedram: OBDD –Based Function Decomposition: Algorithms and Implementation, IEEE Tr. on CAD, vol.15. No 8, Aug.1996, pp. 977-990.

[11] T.Y.Lin and N. Cercone(Eds.): Rough Sets and Data Mining - Analysis of Impresise Data, Kluwer, 1997.

[12] T. Luba: Decomposition of Multiple-Valued Functions, Proc. IEEE ISMVL'95, Bloomington, Indiana, USA, May 23-25, 1995.

[13] T. Luba and J. Rybnik: Rough Sets and Some Aspects of Logic Synthesis, in R. Slowinski (Ed.): Intelligent Decision Support - Handbook of Applications and Advances of the Rough Sets Theory, pp. 181-199, Kluwer, 1993.

[14] R. Murgai, N. Shenoy, R.K. Brayton, A. Sangiovanni Vincentelli: Performance Directed Synthesis for Table Look Up Programmable Gate Arrays, IEEE ICCAD, pp. 572-575, 1991.

[15] R. Murgai, R.K. Brayton, A. Sangiovanni Vincentelli: Optimal Functional Decomposition Using Encoding, 31st ACM/IEEE DAC, pp. 408-414, June 1994.

[16] M. Perkowski, T. Ross, D. Gadd, J.A. Goldman, N. Song: Application of ESOP Minimization in Machine Learning and Knowledge Discover, Proc. Reed Muller'95 Workshop, Chiba, Japan, 1995.

[17] M. Rawski, L. Jozwiak, M. Nowicka, T. Luba: Non-Disjoint Decomposition of Boolean Functions and Its Application in FPGA-oriented Technology Mapping, Proc. of the EUROMICRO'97 Conference, Budapest, Hungary, Sept. 1-4, 1997, pp.24-30, IEEE Computer Society Press.

[18] M.Rawski, L. Jóźwiak, T. Luba, A. Chojnacki: Efficient Logic Synthesis for FPGAs with Functional Decomposition Based on Information Relationship Measures, EUROMICRO'98, Västerås, Sweden, August 25-27, 1998, pp 8-15.

[19] T. Ross, M. Noviskey, T. Taylor, D. Gadd: Pattern Theory: An Engineering Paradigm for Algorithm Design, Final Technical Report, Wright Laboratories, WL/AART/WPAFB, 1991.

[20] F.A.M. Volf, L. Jóźwiak, M.P.J. Stevens: Division-Based versus General Decomposition-Based Multiple-Level Logic Synthesis, VLSI Design, vol.3, No 3, pp. 267 - 287, 1995.

[21] F.A.M. Volf, L. Józwiak: Decompositional Logic Synthesis Approach for Look Up Table Based FPGAs, 8th IEEE International ASIC Conference, Austin, Texas, 18-22 September, 1995.

[22] W. Wan, M.A. Perkowski: A New Approach to the Decomposition of Incompletely specified Multi-Output Functions Based on Graph Coloring and Local Transformation and Its Application to FPGA Mapping, European Design Automation Conference, Hamburg, Germany, September 7-10, 1992, pp. 230-237.

[23] B, Zupan and M. Bohanec: Learning Concept Hierarchies from Examples by Functional Decomposition, Research Report, Department of Inteligent Systems, Josef Stefan Institute, Ljubljana, Slovenia, Sept. 1966.

Probabilistic and Truth-Functional Many-Valued Logic Programming

Thomas Lukasiewicz
Institut für Informatik, Universität Gießen
Arndtstraße 2, D-35392 Gießen, Germany

Abstract

We introduce probabilistic many-valued logic programs in which the implication connective is interpreted as material implication. We show that probabilistic many-valued logic programming is computationally more complex than classical logic programming. More precisely, some deduction problems that are P-complete for classical logic programs are shown to be co-NP-complete for probabilistic many-valued logic programs. We then focus on many-valued logic programming in $\mathrm{Pr}_n^\star$ as an approximation of probabilistic many-valued logic programming. Surprisingly, many-valued logic programs in $\mathrm{Pr}_n^\star$ have both a probabilistic semantics in probabilities over a set of possible worlds and a truth-functional semantics in the finite-valued Łukasiewicz logics $Ł_n$. Moreover, many-valued logic programming in $\mathrm{Pr}_n^\star$ has a model and fixpoint characterization, a proof theory, and computational properties that are very similar to those of classical logic programming.

1. Introduction

We start by presenting probabilistic many-valued logic programs in which the implication connective is interpreted as material implication. We show that probabilistic many-valued logic programming in this framework is computationally more complex than classical logic programming. More precisely, some deduction problems that are P-complete for classical logic programs are shown to be co-NP-complete for probabilistic many-valued logic programs (see also [15] and [14] for other work on the subtleties and the computational complexity of probabilistic deduction).

We then focus on many-valued logic programming in $\mathrm{Pr}_n^\star$ as an approximation of probabilistic many-valued logic programming. Crucially, many-valued logic programming in $\mathrm{Pr}_n^\star$ has a model and fixpoint characterization and a proof theory that are very similar to those of classical logic programming. Furthermore, special cases of many-valued logic programming in $\mathrm{Pr}_n^\star$ have the same computational complexity like their classical counterparts.

Surprisingly (and at first sight even paradoxically), many-valued logic programs in $\mathrm{Pr}_n^\star$ have both a probabilistic semantics in probabilities over a set of possible worlds and a truth-functional semantics in the finite-valued Łukasiewicz logics $Ł_n$. That is, many-valued logic programming in $\mathrm{Pr}_n^\star$ lies in the intersection between probabilistic logics and truth-functional many-valued logics.

The literature already contains quite extensive work on probabilistic and on truth-functional many-valued logic programming separately. However, to the best of our knowledge, an integration of both has never been studied so far.

Probabilistic propositional logics and their various dialects are thoroughly studied in the literature (see, for example, [18] and [5]). Their extensions to probabilistic first-order logics can be classified into first-order logics in which probabilities are defined over a set of possible worlds and those in which probabilities are given over the domain (see, for example, [8] and [2]). The first ones are suitable for representing degrees of belief, while the latter are appropriate for describing statistical knowledge. The same classification holds for probabilistic logic programming (see, for example, [17], [14], and [16]).

Many approaches to truth-functional finite-valued logic programming are restricted to three or four truth values (see, for example, [10], [6], and [4]). Among these approaches, the one closest in spirit to many-valued logic programming in $\mathrm{Pr}_n^\star$ is perhaps the three-valued one in [10].

Many-valued logic programming in $\mathrm{Pr}_n^\star$ is closely related to van Emden's infinite-valued quantitative deduction [19]. More precisely, it is an approximation of probabilistic logic programming under the material implication, while van Emden's quantitative deduction can be understood as an approximation of probabilistic logic programming under the conditional probability implication [14].

Moreover, many-valued logic programming in $\mathrm{Pr}_n^\star$ is related to the work on generalized annotated logic programming [9] and to signed formula logic programming [11].

Many-valued logic programming in $\mathrm{Pr}_n^\star$ itself was initiated in [12], where we already presented a model and fixpoint characterization.

The rest of this paper is organized as follows. Section 2 deals with probabilistic many-valued logic programming. In Section 3, we concentrate on many-valued logic pro-

0-7695-0161-3/99 $10.00 © 1999 IEEE

gramming in $\mathrm{Pr}_n^\star$. Section 4 summarizes the main results.

This paper is an extract from a longer version [13], which includes in full detail all the proofs missing here.

2. Many-valued logic programming in $\mathbf{Pr}_n$

2.1. Technical preliminaries

We briefly summarize how classical first-order logics can be given a probabilistic n-valued semantics with $n \geq 3$ in which probabilities are defined over a set of possible worlds. We basically follow the important work of Halpern [8], which we adapt and restrict to the n-valued setting.

Let $TV = \{0, \frac{1}{n-1}, \frac{2}{n-1}, \ldots, 1\}$ with $n \geq 3$ denote the set of *truth values*. Let Φ be a first-order vocabulary that contains a set of function symbols and a set of predicate symbols (as usual, *constant symbols* are function symbols of arity zero; we say Φ is *function-free* if it does not contain any function symbols of arity greater than zero). Let $\mathcal{X}$ be a set of *object* and *truth variables*.

We define *object terms* by induction as follows. An object term is an object variable from $\mathcal{X}$ or an expression of the kind $f(t_1, \ldots, t_k)$, where f is a function symbol of arity $k \geq 0$ from Φ and $t_1, \ldots, t_k$ are object terms. A *truth term* is a truth value from TV or a truth variable from $\mathcal{X}$. We define *formulas* by induction as follows. If p is a predicate symbol of arity $k \geq 0$ from Φ and $t_1, \ldots, t_k$ are object terms, then $p(t_1, \ldots, t_k)$ is a formula (called *atom*). If F and G are formulas, then $\neg F$, $(F \wedge G)$, $(F \vee G)$, and $(F \leftarrow G)$ are formulas. If F is a formula and x is an object variable from $\mathcal{X}$, then $\forall x\, F$ and $\exists x\, F$ are formulas. An *n-valued formula* is an expression $\mathsf{tv}(F) \geq t$, where F is a classical formula and t is a truth term.

An *interpretation* $I = (D, \pi)$ consists of a nonempty set D, called *domain*, and a mapping π that assigns to each function symbol from Φ a function of right arity over D and to each predicate symbol from Φ a predicate of right arity over D. A *variable assignment* σ is a mapping that assigns to each object variable from $\mathcal{X}$ an element from D and to each truth variable from $\mathcal{X}$ a truth value from TV. For an object variable x from $\mathcal{X}$ and an element d from D, we write $\sigma[x/d]$ to denote the variable assignment that is identical to σ except that it assigns d to x (for a truth variable x from $\mathcal{X}$ and a truth value c from TV, the notation $\sigma[x/c]$ has an analogous meaning). The variable assignment σ is by induction extended to all object and truth terms by defining $\sigma(f(t_1, \ldots, t_k)) = \pi(f)(\sigma(t_1), \ldots, \sigma(t_k))$ for all object terms $f(t_1, \ldots, t_k)$ and $\sigma(c) = c$ for all truth values c from TV. The *truth* of formulas F in I under σ, denoted $I \models_\sigma F$, is inductively defined as follows:

- $I \models_\sigma p(t_1, \ldots, t_k)$ iff $(\sigma(t_1), \ldots, \sigma(t_k)) \in \pi(p)$.
- $I \models_\sigma \neg F$ iff not $I \models_\sigma F$.
- $I \models_\sigma (F \wedge G)$ iff $I \models_\sigma F$ and $I \models_\sigma G$.
- $I \models_\sigma \forall x\, F$ iff $I \models_{\sigma[x/d]} F$ for all $d \in D$.
- The truth of the remaining formulas in I under σ is defined by expressing $\vee$, $\leftarrow$, and $\exists$ in terms of $\neg$, $\wedge$, and $\forall$ as usual.

A formula F *is true* in I, or I is a *model* of F, denoted $I \models F$, iff F is true in I under all variable assignments σ.

A *probabilistic interpretation* (Pr_n*-interpretation*) Pr is a triple $(D, \mathcal{I}, \mu)$, where D is a nonempty set (called *domain*), $\mathcal{I}$ is a set of classical interpretations over D (which are called *possible worlds*) such that $\pi_i(f) = \pi_j(f)$ for all function symbols f from Φ and all interpretations $(D, \pi_i), (D, \pi_j) \in \mathcal{I}$, and μ is a mapping from $\mathcal{I}$ to the set of truth values TV such that all $\mu(I)$ with $I \in \mathcal{I}$ sum up to 1. The *truth value* $Pr_\sigma(F)$ of a formula F in the Pr_n-interpretation Pr under a variable assignment σ is defined as follows (we write $Pr(F)$ if F is variable-free):

$$Pr_\sigma(F) = \textstyle\sum_{I \in \mathcal{I},\, I \models_\sigma F} \mu(I)\,. \qquad (1)$$

An n-valued formula $\mathsf{tv}(F) \geq t$ *is true* in Pr under σ iff $Pr_\sigma(F) \geq \sigma(t)$. An n-valued formula P *is true* in Pr, or Pr is a *model* of P, denoted $Pr \models P$, iff P is true in Pr under all variable assignments σ. Pr is a *model* of a set of n-valued formulas $\mathcal{P}$, denoted $Pr \models \mathcal{P}$, iff Pr is a model of all n-valued formulas in $\mathcal{P}$. $\mathcal{P}$ is *satisfiable* iff a model of $\mathcal{P}$ exists. P is a *logical consequence* of $\mathcal{P}$, denoted $\mathcal{P} \models P$, iff each model of $\mathcal{P}$ is also a model of P.

For an n-valued formula $\mathsf{tv}(F) \geq c$ with a truth value c from TV and a set of n-valued formulas $\mathcal{P}$, let $\boldsymbol{c}$ denote the set of all truth values $Pr_\sigma(F)$ in models Pr of $\mathcal{P}$ under variable assignments σ. It is easy to see that $\mathsf{tv}(F) \geq c$ is a logical consequence of $\mathcal{P}$ iff $c \leq \min \boldsymbol{c}$. Hence, we get a natural notion of tightness for logical consequences: the n-valued formula $\mathsf{tv}(F) \geq c$ is a *tight logical consequence* of $\mathcal{P}$, denoted $\mathcal{P} \models_{\mathit{tight}} \mathsf{tv}(F) \geq c$, iff $c = \min \boldsymbol{c}$.

A *Herbrand* Pr_n*-interpretation* $(\mathcal{I}, \mu)$ consists of a set $\mathcal{I}$ of classical Herbrand interpretations over Φ (that is, subsets of the Herbrand base HB_Φ over Φ) and a mapping μ from $\mathcal{I}$ to TV such that all $\mu(I)$ with $I \in \mathcal{I}$ sum up to 1.

Terms, formulas, n-valued formulas, and sets of n-valued formulas are *ground* iff they do not contain any variables. The notions of substitutions, ground substitutions, instances of formulas, and ground instances of formulas are defined as usual. The last two are assumed to be canonically extended to n-valued formulas. Finally, we also adopt the usual conventions to eliminate parentheses.

2.2. Many-valued logic programs

We now introduce probabilistic many-valued logic programs. We start by defining many-valued program clauses, which are special many-valued formulas.

An *n-valued program clause* is an n-valued formula $\mathsf{tv}(H \vee \neg B_1 \vee \cdots \vee \neg B_k) \geq c$, where $H, B_1, \ldots, B_k$ with

$k \geq 0$ are atoms and c is a truth value from TV. It is abbreviated by $(H \leftarrow B_1, \ldots, B_k)[c, 1]$. Note that all object variables in an n-valued program clause are implicitly universally quantified. An *n-valued logic program* $\mathcal{P}$ is a finite set of n-valued program clauses. We use $ground(\mathcal{P})$ to denote the set of all ground instances of clauses in $\mathcal{P}$.

Many-valued program clauses can be classified into facts and rules: *facts* are of the kind $(H \leftarrow)[c, 1]$, while *rules* have the form $(H \leftarrow B_1, \ldots, B_k)[c, 1]$ with $k > 0$. They can also be divided into logical and purely many-valued program clauses: *logical* program clauses are of the kind $(H \leftarrow B_1, \ldots, B_k)[1, 1]$, while *purely many-valued* ones have the form $(H \leftarrow B_1, \ldots, B_k)[c, 1]$ with $c < 1$.

Next, we introduce many-valued queries, answer substitutions, and answers. An *n-valued query* to an n-valued logic program $\mathcal{P}$ is an expression $\exists(A_1, \ldots, A_l)[t, 1]$, where $A_1, \ldots, A_l$ with $l \geq 1$ are atoms and t is a truth term. An n-valued query is *object-ground* iff it does not contain any object variables. Given an n-valued query $Q_c = \exists(A_1, \ldots, A_l)[c, 1]$ with $c \in TV$, we are interested in its *correct answer substitutions*, which are substitutions θ such that $\mathcal{P} \models \mathsf{tv}((A_1 \wedge \cdots \wedge A_l)\theta) \geq c$ and that θ acts only on variables in Q_c. The *correct answer* for Q_c is Yes if a correct answer substitution exists and No otherwise. Given an n-valued query $Q_x = \exists(A_1, \ldots, A_l)[x, 1]$ with $x \in \mathcal{X}$, we are interested in its *tight answer substitutions*, which are substitutions θ such that $\mathcal{P} \models_{tight} \mathsf{tv}((A_1 \wedge \cdots \wedge A_l)\theta) \geq x\theta$, that θ acts only on variables in Q_x, and that $x\theta$ is a truth value from TV. Note that such n-valued queries Q_x always have a tight answer substitution.

Example 2.1 Let $n = 101$ and let the n-valued logic program $\mathcal{P}$ contain the following rules and facts (R, S, and T are object variables; h, a, b, and o are constants):

$$(re(R, S) \leftarrow ro(R, S))[.7, 1]$$
$$(re(R, S) \leftarrow ro(R, S), so(R, S))[.9, 1]$$
$$(re(R, S) \leftarrow ro(R, S), ad(R, S))[1, 1]$$
$$(re(R, S) \leftarrow re(R, T), re(T, S))[1, 1]$$
$$(ro(h, a) \leftarrow)[1, 1], (ad(h, a) \leftarrow)[1, 1]$$
$$(ro(a, b) \leftarrow)[1, 1], (ad(a, b) \leftarrow)[.8, 1]$$
$$(ro(b, o) \leftarrow)[1, 1], (so(b, o) \leftarrow)[1, 1]$$

Then, some many-valued queries are $\exists(re(h, o))[.99, 1]$, $\exists(re(h, U))[.8, 1]$, and $\exists(re(h, o))[X, 1]$, where U is an object variable and X is a truth variable. The correct answer for $\exists(re(h, o))[.99, 1]$ to $\mathcal{P}$ is No, whereas the correct answer for $\exists(re(h, U))[.8, 1]$ to $\mathcal{P}$ is Yes (all the correct answer substitutions for $\exists(re(h, U))[.8,1]$ to $\mathcal{P}$ are given by $\{U/a\}$ and $\{U/b\}$). Finally, the unique tight answer substitution for $\exists(re(h, o))[X, 1]$ to $\mathcal{P}$ is given by $\{X/.7\}$.

Like classical logic programs, many-valued logic programs have the nice property that they are always satisfiable [13]. Furthermore, ground many-valued formulas are logically entailed in Pr_n-interpretations iff they are logically entailed in Herbrand Pr_n-interpretations [13].

In the sequel, we use *probabilistic many-valued logic programming* as a synonym for the problem of deciding whether Yes is the correct answer for a given ground many-valued query to a many-valued logic program.

2.3. Computational complexity

We now analyze the computational complexity of two decidable special cases of probabilistic many-valued logic programming. The first one is a generalization of propositional logic programming, while the second one generalizes the decision problem that defines the data complexity of datalog. These two special cases are of special interest, since their classical counterparts have the nice property that they are P-complete (see, for example, [3] for a survey).

Crucially, the P-completeness does not carry over to the two probabilistic many-valued generalizations:

Theorem 2.2 *a) The problem of deciding whether* Yes *is the correct answer for a ground n-valued query $\exists(A_1, \ldots, A_l)[c, 1]$ to a ground n-valued logic program $\mathcal{P}$ is co-NP-complete. b) Let Φ be function-free. Let $\mathcal{P}$ be a fixed n-valued logic program and let $\mathcal{F}$ be a varying finite set of ground logical facts. Let $\mathcal{P} \cup \mathcal{F}$ contain all constant symbols from Φ. The problem of deciding whether* Yes *is the correct answer for a ground n-valued query $\exists(A_1, \ldots, A_l)[c, 1]$ to $\mathcal{P} \cup \mathcal{F}$ is co-NP-complete.*

Hence, restricted deduction problems that are computationally tractable for classical logic programs are presumably intractable for many-valued logic programs. Thus, any attempt towards efficient probabilistic many-valued logic programming should be guided by looking for efficient special-case, average-case, or approximation techniques.

3. Many-valued logic programming in $\mathrm{Pr}_n^\star$

3.1. $\mathrm{Pr}_n^\star$-interpretations

Probabilistic many-valued logic programming as introduced in Section 2.2 has a well-defined probabilistic semantics. However, its increased computational complexity compared to classical logic programming is quite discouraging for a broad use in practice, especially for a possible application in large knowledge-base systems.

This increase in complexity seems to be mainly due to the probabilistic semantics in its full generality. In fact, we now provide a truth-functional approach to many-valued logic programming that approximates our probabilistic one and that is less computationally complex. The main idea is to focus on a special kind of Pr_n-interpretations:

A $\mathrm{Pr}_n^\star$-*interpretation* is a Pr_n-interpretation Pr with

$$Pr_\sigma(A \wedge B) = \min(Pr_\sigma(A), Pr_\sigma(B)) \qquad (2)$$

for all variable assignments σ and all atoms A and B.

Interestingly, (2) is equivalently expressed as follows.

Theorem 3.1 *Let $Pr = (D, \mathcal{I}, \mu)$ be a Pr_n-interpretation. It holds $Pr_\sigma(A \wedge B) = \min(Pr_\sigma(A), Pr_\sigma(B))$ for all variable assignments σ and all atoms A and B iff all the interpretations $I \in \mathcal{I}$ with $\mu(I) > 0$ can be written in a sequence $(D, \pi_1), \ldots, (D, \pi_k)$ such that for all predicate symbols p from Φ: $\pi_1(p) \supseteq \pi_2(p) \supseteq \cdots \supseteq \pi_k(p)$.*

A $\mathrm{Pr}_n^\star$-*model* of a set of n-valued formulas $\mathcal{P}$ is a $\mathrm{Pr}_n^\star$-interpretation that is a model of $\mathcal{P}$. The set of n-valued formulas $\mathcal{P}$ is *satisfiable in* $\mathrm{Pr}_n^\star$ iff a $\mathrm{Pr}_n^\star$-model of $\mathcal{P}$ exists. The n-valued formula P is a *logical consequence in* $\mathrm{Pr}_n^\star$ of $\mathcal{P}$ iff each $\mathrm{Pr}_n^\star$-model of $\mathcal{P}$ is also a model of P. The n-valued formula $\mathrm{tv}(F) \geq c$ is a *tight logical consequence in* $\mathrm{Pr}_n^\star$ of $\mathcal{P}$ iff c is the minimum of all truth values $Pr_\sigma(F)$ in $\mathrm{Pr}_n^\star$-models Pr of $\mathcal{P}$ under variable assignments σ.

The next theorem shows that tight logical consequences in $\mathrm{Pr}_n^\star$ approximate logical and tight logical consequences in Pr_n. In particular, for many-valued logic programs $\mathcal{P}$ and formulas F, this theorem shows that $\mathcal{P} \models_{tight} \mathrm{tv}(F) \geq 0$ in $\mathrm{Pr}_n^\star$ immediately entails $\mathcal{P} \models_{tight} \mathrm{tv}(F) \geq 0$ in Pr_n.

Theorem 3.2 *Let $\mathcal{F}$ be a set of n-valued formulas, let F be a formula, and let $c \in TV$. If $\mathcal{F} \models_{tight} \mathrm{tv}(F) \geq c$ in $\mathrm{Pr}_n^\star$, then all truth values $d \in TV$ with $\mathcal{F} \models \mathrm{tv}(F) \geq d$ in Pr_n are contained in $\{0, \ldots, c\} \subseteq TV$.*

3.2. Comparison with $\mathrm{Ł}_n$-interpretations

We now focus on the relationship between $\mathrm{Pr}_n^\star$-interpretations and interpretations in $\mathrm{Ł}_n$. We first define $\mathrm{Ł}_n$-interpretations and the truth value of classical formulas in $\mathrm{Ł}_n$-interpretations under variable assignments.

An $\mathrm{Ł}_n$-*interpretation* $L = (D, \pi)$ consists of a non-empty domain D and a mapping π that assigns to each k-ary function symbol from Φ a mapping from D^k to D and to each k-ary predicate symbol from Φ a mapping from D^k to the set of truth values TV. The *truth value* $L_\sigma(F)$ of a formula F in the $\mathrm{Ł}_n$-interpretation L under a variable assignment σ is inductively defined by:

- $L_\sigma(p(t_1, \ldots, t_k)) = \pi(p)(\sigma(t_1), \ldots, \sigma(t_k))$.
- $L_\sigma(\neg F) = 1 - L_\sigma(F)$.
- $L_\sigma(F \wedge G) = \min(L_\sigma(F), L_\sigma(G))$.
- $L_\sigma(F \vee G) = \max(L_\sigma(F), L_\sigma(G))$.
- $L_\sigma(F \leftarrow G) = \min(1, L_\sigma(F) - L_\sigma(G) + 1)$.
- $L_\sigma(\forall x\, F) = \min\{L_{\sigma[x/d]}(F) \mid d \in D\}$.
- $L_\sigma(\exists x\, F) = \max\{L_{\sigma[x/d]}(F) \mid d \in D\}$.

We next show that for logical combinations of certain formulas, the truth value in $\mathrm{Pr}_n^\star$-interpretations under variable assignments is defined like the truth value in $\mathrm{Ł}_n$-interpretations under variable assignments.

Lemma 3.3 *Let $Pr = (D, \mathcal{I}, \mu)$ be a $\mathrm{Pr}_n^\star$-interpretation and let σ be a variable assignment. For all object variables $x \in \mathcal{X}$, all formulas F, and all formulas G and H that are built without the logical connectives $\neg$ and $\leftarrow$:*

$$Pr_\sigma(\neg F) = 1 - Pr_\sigma(F) \qquad (3)$$
$$Pr_\sigma(G \wedge H) = \min(Pr_\sigma(G), Pr_\sigma(H)) \qquad (4)$$
$$Pr_\sigma(G \vee H) = \max(Pr_\sigma(G), Pr_\sigma(H)) \qquad (5)$$
$$Pr_\sigma(G \leftarrow H) = \min(1, Pr_\sigma(G) - Pr_\sigma(H) + 1) \qquad (6)$$
$$Pr_\sigma(\forall x\, G) = \min\{Pr_{\sigma[x/d]}(G) \mid d \in D\} \qquad (7)$$
$$Pr_\sigma(\exists x\, G) = \max\{Pr_{\sigma[x/d]}(G) \mid d \in D\}. \qquad (8)$$

This means that $\mathrm{Pr}_n^\star$- and $\mathrm{Ł}_n$-interpretations give the same truth value to all formulas built without the logical connectives $\neg$ and $\leftarrow$, and to all logical combinations of these formulas (thus, also to classical program clauses):

Theorem 3.4 *Let Pr be a $\mathrm{Pr}_n^\star$-interpretation, let L be an $\mathrm{Ł}_n$-interpretation, and let σ be a variable assignment. If $Pr_\sigma(A) = L_\sigma(A)$ for all atoms A, then $Pr_\sigma(G) = L_\sigma(G)$, $Pr_\sigma(\neg G) = L_\sigma(\neg G)$, and $Pr_\sigma(G \leftarrow H) = L_\sigma(G \leftarrow H)$ for all formulas G and H built without $\neg$ and $\leftarrow$.*

Note that there also exist formulas with different truth values in $\mathrm{Pr}_n^\star$-interpretations and in $\mathrm{Ł}_n$-interpretations:

Theorem 3.5 *There are $\mathrm{Pr}_n^\star$-interpretations Pr, $\mathrm{Ł}_n$-interpretations L, variable assignments σ, and formulas G with $Pr_\sigma(A) = L_\sigma(A)$ for all atoms A and $Pr_\sigma(G) \neq L_\sigma(G)$.*

This last theorem is not surprising, since $\mathrm{Pr}_n^\star$-interpretations still satisfy the axioms of probability. That is, $\mathrm{Pr}_n^\star$-interpretations always give the same truth value to formulas that are logically equivalent in the classical sense. $\mathrm{Ł}_n$-interpretations, in contrast, do not have this property.

3.3. Many-valued logic programs

We keep the definitions of many-valued program clauses and many-valued programs from Section 2.2. In particular, the semantics of many-valued program clauses in $\mathrm{Pr}_n^\star$-interpretations is already given by the semantics of many-valued formulas in Pr_n-interpretations. The truth of many-valued program clauses in $\mathrm{Pr}_n^\star$-interpretations is then additionally characterized as follows.

Lemma 3.6 *For all $\mathrm{Pr}_n^\star$-interpretations $Pr = (D, \mathcal{I}, \mu)$, all variable assignments σ, and all n-valued program clauses $(H \leftarrow B_1, \ldots, B_k)[c, 1]$:*

$$(H \leftarrow B_1, \ldots, B_k)[c, 1] \text{ is true in } Pr \text{ under } \sigma \text{ iff}$$
$$Pr_\sigma(H) \geq c - 1 + \min(Pr_\sigma(B_1), \ldots, Pr_\sigma(B_k)).$$

Given an n-valued query $Q_c = \exists(A_1, \ldots, A_l)[c, 1]$ with $c \in TV$, we are interested in its *correct answer substitutions in* $\mathrm{Pr}_n^\star$, which are substitutions θ such that $\mathcal{P} \models \mathrm{tv}(A_1\theta \wedge \cdots \wedge A_l\theta) \geq c$ in $\mathrm{Pr}_n^\star$ and that θ acts only on variables in Q_c. The *correct answer in* $\mathrm{Pr}_n^\star$ for Q_c is Yes if a correct answer substitution in $\mathrm{Pr}_n^\star$ exists and No otherwise. Given an n-valued query $Q_x = \exists(A_1, \ldots, A_l)[x, 1]$ with $x \in \mathcal{X}$, we are interested in its *tight answer substitutions in* $\mathrm{Pr}_n^\star$, which are substitutions θ such that $\mathcal{P} \models_{tight} \mathrm{tv}(A_1\theta \wedge \cdots \wedge A_l\theta) \geq x\theta$ in $\mathrm{Pr}_n^\star$, that θ acts only on variables in Q_x, and that $x\theta$ is a truth value from TV.

Example 3.7 Let $n = 101$ and let $\mathcal{P}$ be the n-valued logic program from Example 2.1. The correct answer in $\mathrm{Pr}_n^\star$ for the n-valued query $\exists(re(h, o))[.99, 1]$ to $\mathcal{P}$ is No, whereas the correct answer in $\mathrm{Pr}_n^\star$ for $\exists(re(h, U))[.8, 1]$ to $\mathcal{P}$ is Yes (note that all the correct answer substitutions in $\mathrm{Pr}_n^\star$ for $\exists(re(h,U))[.8,1]$ to $\mathcal{P}$ are given by $\{U/a\}$, $\{U/b\}$, and $\{U/o\}$). Finally, the unique tight answer substitution in $\mathrm{Pr}_n^\star$ for $\exists(re(h, o))[X, 1]$ to $\mathcal{P}$ is given by $\{X/.8\}$.

Note that many-valued logic programs are always satisfiable in $\mathrm{Pr}_n^\star$ [13]. Moreover, ground many-valued formulas are logically entailed in $\mathrm{Pr}_n^\star$-interpretations iff they are logically entailed in Herbrand $\mathrm{Pr}_n^\star$-interpretations [13].

In the sequel, we use *many-valued logic programming in* $\mathrm{Pr}_n^\star$ as a synonym for the problem of deciding whether Yes is the correct answer in $\mathrm{Pr}_n^\star$ for a given ground many-valued query to a many-valued logic program.

3.4. Model and fixpoint semantics

We briefly discuss the model and fixpoint semantics of many-valued logic programs in $\mathrm{Pr}_n^\star$ [12]. In the sequel, let $\mathcal{P}$ be an n-valued logic program.

We focus on Herbrand $\mathrm{Pr}_n^\star$-interpretations, which we identify with fuzzy sets. In detail, each Herbrand $\mathrm{Pr}_n^\star$-interpretation $(\mathcal{I}, \mu)$ is identified with the fuzzy set $\boldsymbol{I}$: $HB_\Phi \to TV$, where $\boldsymbol{I}[A]$, for all $A \in HB_\Phi$, is the sum of all $\mu(I)$ with $I \in \mathcal{I}$ and $I \models A$. We subsequently use bold symbols to denote such fuzzy sets. The fuzzy sets $\emptyset$ and $\boldsymbol{HB}_\Phi$ are defined by $\emptyset[A] = 0$ and $\boldsymbol{HB}_\Phi[A] = 1$ for all $A \in HB_\Phi$. Finally, we define the intersection, the union, and the subset relation for fuzzy sets $\boldsymbol{S}_1$ and $\boldsymbol{S}_2$ as usual by $\boldsymbol{S}_1 \cap \boldsymbol{S}_2 = \min(\boldsymbol{S}_1, \boldsymbol{S}_2)$, $\boldsymbol{S}_1 \cup \boldsymbol{S}_2 = \max(\boldsymbol{S}_1, \boldsymbol{S}_2)$, and $\boldsymbol{S}_1 \subseteq \boldsymbol{S}_2$ iff $\boldsymbol{S}_1 = \boldsymbol{S}_1 \cap \boldsymbol{S}_2$, respectively.

We define the immediate consequence operator $\boldsymbol{T}_\mathcal{P}$ as follows. For all $\boldsymbol{I} \subseteq \boldsymbol{HB}_\Phi$ and $H \in HB_\Phi$:

$$\boldsymbol{T}_\mathcal{P}(\boldsymbol{I})[H] = \max(\{c - 1 + \min(\boldsymbol{I}[B_1], \ldots, \boldsymbol{I}[B_k]) \mid (H \leftarrow B_1, \ldots, B_k)[c, 1] \in \mathit{ground}(\mathcal{P})\} \cup \{0\}).$$

Note that we define $\min(\boldsymbol{I}[B_1], \ldots, \boldsymbol{I}[B_k]) = 1$ for $k = 0$.

For all $\boldsymbol{I} \subseteq \boldsymbol{HB}_\Phi$, we define $\boldsymbol{T}_\mathcal{P}{\uparrow}\omega(\boldsymbol{I})$ as the union of all $\boldsymbol{T}_\mathcal{P}{\uparrow}l(\boldsymbol{I})$ with $l < \omega$, where $\boldsymbol{T}_\mathcal{P}{\uparrow}0(\boldsymbol{I}) = \boldsymbol{I}$ and $\boldsymbol{T}_\mathcal{P}{\uparrow}(l+1)(\boldsymbol{I}) = \boldsymbol{T}_\mathcal{P}(\boldsymbol{T}_\mathcal{P}{\uparrow}l(\boldsymbol{I}))$ for all $l < \omega$. Finally, we abbreviate $\boldsymbol{T}_\mathcal{P}{\uparrow}\alpha(\emptyset)$ by $\boldsymbol{T}_\mathcal{P}{\uparrow}\alpha$.

The model and fixpoint semantics of many-valued logic programs in $\mathrm{Pr}_n^\star$ is now expressed as follows.

Theorem 3.8

$$\bigcap\{\boldsymbol{I} \mid \boldsymbol{I} \subseteq \boldsymbol{HB}_\Phi, \boldsymbol{I} \models \mathcal{P}\} = \mathit{lfp}(\boldsymbol{T}_\mathcal{P}) = \boldsymbol{T}_\mathcal{P}{\uparrow}\omega.$$

Thus, tight answer substitutions for object-ground many-valued queries can be characterized as follows.

Theorem 3.9 *Let $\mathcal{P}$ be an n-valued logic program and let $\exists(A_1, \ldots, A_l)[x, 1]$ be an object-ground n-valued query with $x \in \mathcal{X}$. The tight answer substitution in $\mathrm{Pr}_n^\star$ for $\exists(A_1, \ldots, A_l)[x, 1]$ to $\mathcal{P}$ is given by $\{x/c\}$, where c is the minimum of all $\boldsymbol{T}_\mathcal{P}{\uparrow}\omega[A_i]$ with $i \in [1{:}l]$.*

3.5. Proof theory

We now present SLD$\mathrm{Pr}_n^\star$-resolution for many-valued logic programs in $\mathrm{Pr}_n^\star$, which is an extension of the classical SLD-resolution (see, for example, [1]). In the sequel, many-valued facts $(A \leftarrow)[c, 1]$ are abbreviated by $(A)[c, 1]$.

A *subgoal list* is a finite list $(A_1)[a_1, 1] \ldots (A_m)[a_m, 1]$ of n-valued facts $(A_1)[a_1, 1], \ldots, (A_m)[a_m, 1]$ such that $a_1, \ldots, a_m > 0$ and $m \geq 0$. A substitution θ is applied to a subgoal list by replacing each contained atom A_i by $A_i\theta$. For n-valued program clauses P_1 and P_2, we say P_1 is a *variant* of P_2 iff P_1 is an instance of P_2 and P_2 is an instance of P_1. The notions of unifiers and most general unifiers (mgu) are defined as usual.

The subgoal list $(\alpha(B_1)[b, 1] \ldots (B_k)[b, 1]\omega)\theta$ is a *resolvent* of the subgoal list $\alpha(A)[a, 1]\omega$ and the n-valued program clause $(H \leftarrow B_1, \ldots, B_k)[c, 1]$ *with* mgu θ iff A and H unify with mgu θ, $a \leq c$, and $b = a - c + 1$.

Note that, for subgoal lists $\alpha(A)[a, 1]\omega$ and n-valued program clauses $(H \leftarrow B_1, \ldots, B_k)[c, 1]$, the resolvent $(\alpha(B_1)[b, 1] \ldots (B_k)[b, 1]\omega)\theta$ is a subgoal list, since $0 < a \leq c \leq 1$ and $b = a - c + 1$ entails $0 < b \leq 1$.

An SLD$\mathrm{Pr}_n^\star$*-derivation* of a subgoal list R_0 from an n-valued logic program $\mathcal{P}$ is a maximal sequence R_0, $(C_0, \theta_0), R_1, (C_1, \theta_1), \ldots$, where $R_0, R_1, \ldots$ is a sequence of subgoal lists, $C_0, C_1, \ldots$ is a sequence of variants of clauses from $\mathcal{P}$, and $\theta_0, \theta_1, \ldots$ is a sequence of substitutions such that R_{i+1} is a resolvent of R_i and C_i with mgu θ_i and such that C_i does not have any variables in common with $R_0, C_0, \ldots, R_{i-1}$. If a subgoal list R_j is empty, then it is the last one in a derivation. Such an SLD$\mathrm{Pr}_n^\star$-derivation is called *successful*.

The presented SLD$\mathrm{Pr}_n^\star$-resolution is a sound and complete technique for correct query answering in $\mathrm{Pr}_n^\star$. That is, for n-valued logic programs $\mathcal{P}$ and n-valued queries $Q_c = \exists(A_1, \ldots, A_l)[c,1]$ with $c > 0$, the correct answer in $\mathrm{Pr}_n^\star$ for Q_c to $\mathcal{P}$ is Yes iff a successful SLD$\mathrm{Pr}_n^\star$-derivation of $(A_1)[c, 1] \ldots (A_l)[c, 1]$ from $\mathcal{P}$ exists. Moreover, each successful SLD$\mathrm{Pr}_n^\star$-derivation of $(A_1)[c, 1] \ldots (A_l)[c, 1]$ from

$\mathcal{P}$ with the sequence of substitutions $\theta_0, \theta_1, \ldots, \theta_j$ provides a correct answer substitution in $\mathrm{Pr}_n^\star$ for Q_c to $\mathcal{P}$ by the substitution $\theta_0\theta_1 \ldots \theta_j$ restricted to the variables in Q_c.

More precisely, the soundness and the completeness of $\mathrm{SLDPr}_n^\star$-resolution is expressed as follows.

Theorem 3.10 *a) Let $\mathcal{P}$ be an n-valued logic program and $Q_c = \exists(A_1, \ldots, A_l)[c, 1]$ be an n-valued query with $c > 0$. If there exists a successful $\mathrm{SLDPr}_n^\star$-derivation of $(A_1)[c, 1] \ldots (A_l)[c, 1]$ from $\mathcal{P}$ with the sequence of substitutions $\theta_0, \theta_1, \ldots, \theta_j$, then the substitution $\theta_0\theta_1 \ldots \theta_j$ restricted to the variables in Q_c is a correct answer substitution in $\mathrm{Pr}_n^\star$ for Q_c to $\mathcal{P}$. b) Let $\mathcal{P}$ be an n-valued logic program and $Q_c = \exists(A_1, \ldots, A_l)[c,1]$ be an n-valued query with $c > 0$. If* Yes *is the correct answer in $\mathrm{Pr}_n^\star$ for Q_c to $\mathcal{P}$, then a successful $\mathrm{SLDPr}_n^\star$-derivation of $(A_1)[c, 1] \ldots (A_l)[c, 1]$ from $\mathcal{P}$ exists.*

3.6. Computational complexity

We now focus on the computational complexity of many-valued logic programming in $\mathrm{Pr}_n^\star$. Like in Section 2.3, we concentrate on the two decidable special cases that generalize propositional logic programming and the decision problem that defines the data complexity of datalog. Crucially, in contrast to the probabilistic many-valued generalizations, the truth-functional ones are P-complete.

Theorem 3.11 *a) The optimization problem of computing the tight answer substitution in $\mathrm{Pr}_n^\star$ for an object-ground n-valued query $\exists(A_1, \ldots, A_l)[x, 1]$, with $x \in \mathcal{X}$, to a ground n-valued logic program $\mathcal{P}$ is P-complete. b) Let Φ be function-free. Let $\mathcal{P}$ be a fixed n-valued logic program, let $\mathcal{F}$ be a varying finite set of ground n-valued facts. Let $\mathcal{P} \cup \mathcal{F}$ contain all constant symbols from Φ. The optimization problem of computing the tight answer substitution in $\mathrm{Pr}_n^\star$ for an object-ground n-valued query $\exists(A_1, \ldots, A_l)[x, 1]$, with $x \in \mathcal{X}$, to $\mathcal{P} \cup \mathcal{F}$ is P-complete.*

4. Summary and conclusion

We introduced probabilistic many-valued logic programs in which the implication connective is interpreted as material implication. We showed that probabilistic many-valued logic programming is computationally more complex than classical logic programming. We then focused on the approximation of probabilistic many-valued logic programming by many-valued logic programming in $\mathrm{Pr}_n^\star$. In particular, we introduced a sound and complete proof theory for many-valued logic programming in $\mathrm{Pr}_n^\star$.

Crucially, many-valued logic programs in $\mathrm{Pr}_n^\star$ have both a probabilistic semantics in probabilities over a set of possible worlds and a truth-functional semantics in the finite-valued Łukasiewicz logics $Ł_n$. Furthermore, many-valued logic programming in $\mathrm{Pr}_n^\star$ has a model and fixpoint characterization, a proof theory, and computational properties that are very similar to those of classical logic programming. Hence, it is well worth being studied more deeply.

Finally, this paper showed how presumably intractable probabilistic deduction problems in artificial intelligence can be tackled by efficient approximation techniques based on truth-functional many-valued logics.

References

[1] K. R. Apt. Logic programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 10, pages 493–574. MIT Press, 1990.

[2] F. Bacchus, A. Grove, J. Y. Halpern, and D. Koller. From statistical knowledge bases to degrees of beliefs. *Artif. Intell.*, 87:75–143, 1996.

[3] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. In *Proc. of the 12th Annual IEEE Conference on Computational Complexity*, pages 82–101, 1997.

[4] J. P. Delahaye and V. Thibau. Programming in three-valued logic. *Theor. Comput. Sci.*, 78:189–216, 1991.

[5] R. Fagin, J. Y. Halpern, and N. Megiddo. A logic for reasoning about probabilities. *Inf. Comput.*, 87:78–128, 1990.

[6] M. Fitting. Bilattices and the semantics of logic programming. *J. Log. Program.*, 11(1–2):91–116, 1991.

[7] R. Hähnle and G. Escalada-Imaz. Deduction in many-valued logics: a survey. *Mathware Soft Comput.*, IV(2):69–97, 1997.

[8] J. Y. Halpern. An analysis of first-order logics of probability. *Artif. Intell.*, 46:311–350, 1990.

[9] M. Kifer and V. S. Subrahmanian. Theory of generalized annotated logic programming and its applications. *J. Log. Program.*, 12(3–4):335–367, 1992.

[10] J.-L. Lassez and M. J. Maher. Optimal fixedpoints of logic programs. *Theor. Comput. Sci.*, 39:15–25, 1985.

[11] J. J. Lu. Logic programming with signs and annotations. *J. Log. Comput.*, 6(6):755–778, 1996.

[12] T. Lukasiewicz. Many-valued first-order logics with probabilistic semantics. In *Proc. of the Annual Conference of the European Association for Computer Science Logic*, 1998.

[13] T. Lukasiewicz. Probabilistic and truth-functional many-valued logic programming. Technical Report 9809, Institut für Informatik, Universität Gießen, 1998.

[14] T. Lukasiewicz. Probabilistic logic programming. In *Proc. of the 13th Biennial European Conference on Artificial Intelligence*, pages 388–392. J. Wiley & Sons, 1998.

[15] T. Lukasiewicz. Local probabilistic deduction from taxonomic and probabilistic knowledge-bases over conjunctive events. *Int. J. Approx. Reasoning*, 1999. To appear.

[16] R. T. Ng. Semantics, consistency, and query processing of empirical deductive databases. *IEEE Trans. Knowl. Data Eng.*, 9(1):32–49, 1997.

[17] R. T. Ng and V. S. Subrahmanian. A semantical framework for supporting subjective and conditional probabilities in deductive databases. *J. Autom. Reasoning*, 10(2):191–235, 1993.

[18] N. J. Nilsson. Probabilistic logic. *Artif. Intell.*, 28:71–88, 1986.

[19] M. H. van Emden. Quantitative deduction and its fixpoint theory. *J. Log. Program.*, 3(1):37–53, 1986.

Representation Theorems and Theorem Proving in Non-Classical Logics

Viorica Sofronie-Stokkermans
Max-Planck-Institut für Informatik, D-66123 Saarbrücken, Germany
email: sofronie@mpi-sb.mpg.de

Abstract

In this paper we present a method for automated theorem proving in non-classical logics having as algebraic models bounded distributive lattices with certain types of operators. The idea is to use a Priestley-style representation for distributive lattices with operators in order to define a class of Kripke-style models with respect to which the logic is sound and complete. If this class of Kripke-style models is elementary, it can then be used for a translation to clause form; satisfiability of the resulting clauses can be checked by resolution. We illustrate the ideas by several examples.

1 Introduction

Efficient reasoning on incomplete, vague and imprecise knowledge requires the development of efficient many-valued theorem provers. Since many non-classical logics that occur in a natural way in practical applications can be proved to be sound and complete with respect to certain classes of distributive lattices with operators, in this paper we will focus on this kind of logics. One of the main advantages of distributive lattices (with well-behaved operators) is that they usually have good and economical representation theorems. The method for automated theorem proving presented here uses the Priestley representation for distributive lattices with operators. We illustrate the main ideas by several examples, one of which is presented in detail.

The main contributions of this paper are the following:

- A link between algebraic and Kripke-style models for logics based on distributive lattices with operators, established by using Priestley-style representation theorems. This extends the results in [21], and makes more precise some results from [24].
- A method for translation to clause form in logics based on distributive lattices with operators.
- Decidability results for some of the logics considered.

Our work is inspired by the work of Baaz and Fermüller [1] on many-valued resolution, by the results of Hähnle [7, 8], and by our previous work on finitely-valued logics [21] on the one hand; and by the results of Goldblatt [5] on Priestley-type dualities for distributive algebras with operators on the other hand. The method presented in this paper can be seen as a common framework which encompasses existing results in automated theorem proving in both regular finitely-valued logics, and modal logics.

The paper is structured as follows. In Section 2 we give some background information. In Section 3 we briefly present a Priestley representation for distributive lattices with operators, and its use in establishing links between algebraic and Kripke-style models. In Section 4 we describe a method for translation to clause form and automated theorem proving. In Section 5 we give several examples, one of which is presented in detail. We end by mentioning related approaches and plans for future work.

2 Preliminaries

Algebra. In what follows we assume known standard notions, such as partially-ordered set, lattice, order-filter and order-ideal in partially-ordered sets, meet- and join-irreducible elements, and (prime) filter and (prime) ideal in lattices. For definitions and further information we refer to [3]. Let $(X, \leq)$ be a partially ordered set and let $R \subseteq X^{n+1}$ be a $(n+1)$-ary relation on X. R is an *increasing relation* if it has the property that for all $\overline{x} \in X^n$ and every $y, z \in X$, if $R(\overline{x}, y)$ and $y \leq z$ then $R(\overline{x}, z)$. R is a *decreasing relation* if for all $\overline{x} \in X^n$ and every $y, z \in X$, if $R(\overline{x}, y)$ and $z \leq y$ then $R(\overline{x}, z)$. Given a partially-ordered set $(X, \leq)$, $\mathcal{O}(X)$ will denote the set of order-filters of X. For the basic notions of universal algebra needed in what follows, such as satisfiability of equations in an algebra or a class of algebras (notation: $A \stackrel{a}{\models} t_1 = t_2$ resp. $\mathcal{K} \stackrel{a}{\models} t_1 = t_2$) and the definition of a variety, we refer to [2]. The variety generated by a class $\mathcal{K}$ of algebras will be denoted $HSP(\mathcal{K})$.

Logic. Roughly speaking, a logic is described by a pair $\mathcal{L} = (\mathsf{Fma}(\mathcal{L}), \vdash_{\mathcal{L}})$, where $\mathsf{Fma}(\mathcal{L})$ is the set of formulae in the logic and $\vdash_{\mathcal{L}}$ is a relation between sets of formulae and formulae called *provability relation* (for simplicity we only consider here the propositional case). A formula ϕ is a

0-7695-0161-3/99 $10.00 © 1999 IEEE

theorem of $\mathcal{L}$ (denoted $\vdash_{\mathcal{L}} \phi$) if $\emptyset \vdash_{\mathcal{L}} \phi$. We call a logic $\mathcal{L}$ *sound* w.r.t. a class $\mathcal{K}$ of models if every theorem ϕ of $\mathcal{L}$ is true in all models in $\mathcal{K}$; we call $\mathcal{L}$ *complete* w.r.t. $\mathcal{K}$ if every formula ϕ that is true in all models in $\mathcal{K}$ is a theorem of $\mathcal{L}$.

3 Priestley representation

We first present some results on Priestley representation for distributive lattices with operators (abbreviated in what follows DLO). For details we refer to [3, 5, 21, 24].

The Priestley representation theorem [17] states that every bounded distributive lattice A is isomorphic to the lattice of clopen (i.e. closed and open) order filters of the ordered topological space having as points the prime filters of A, ordered by inclusion, and the topology generated by the sets of the form $X_a = \{F \mid F \text{ prime filter}, a \in F\}$ and their complements as a subbasis. We denote the partially ordered set of all prime filters of A, ordered by inclusion, and endowed with the topology mentioned above, by $D(A)$ (we will refer to it as the dual of A). The lattice of clopen order filters of an ordered topological space $X = (X, \leq, \tau)$ will be denoted by $E(X)$. The Priestley representation theorem states that every bounded distributive lattice A is isomorphic to $E(D(A))$.

In this paper we will also consider other classes of operators on bounded distributive lattices, besides $\vee$ and $\wedge$, as explained in what follows. Let Σ be a signature containing function symbols in several classes. In order to distinguish these classes, we will write $\Sigma = Lh \cup La \cup Jh \cup Mh \cup Ja \cup Ma$, where Lh, La, Jh, Mh, Ja, and Ma may be empty. Let DLO_Σ be the class of all bounded distributive lattices with operators in Σ, such that for every $A \in \mathsf{DLO}_\Sigma$ the operators in the classes Lh, La, Jh, Mh, Ja, and Ma are respectively lattice morphisms and antimorphisms, join and meet hemimorphisms (maps of arbitrary arity, preserving finite joins resp. meets in every argument), and join resp. meet hemiantimorphisms (maps of arbitrary arity, mapping finite meets to joins (resp. joins to meets) in every argument). The main fact proved in [24] is that the additional operators in Σ on an algebra $A \in \mathsf{DLO}_\Sigma$ induce functions resp. relations on $D(A)$, which, again, define corresponding operators on both the set $E(D(A))$ of clopen order-filters of $D(A)$ and on the set $\mathcal{O}(D(A))$ of order-filters of $D(A)$. For details on a Priestley duality theorem for DLO we refer to [24]. The main result we will use here is that for every $A \in \mathsf{DLO}_\Sigma$, there exists an isomorphism of $\{0, 1, \vee, \wedge\} \cup \Sigma$-algebras $\eta_A : A \simeq E(D(A))$. Note that if A is finite then the Priestley topology on $D(A)$ is discrete, hence $E(D(A)) = \mathcal{O}(D(A))$.

In [21, 23] we used this Priestley representation for DLO to give an automated theorem proving procedure for certain finitely-valued logics. In this paper we show that the ideas are much more general, and can also be applied, with minor modifications, to wider classes of non-classical logics.

3.1 Kripke-style models

In [24], inspired by the definition of the relational Priestley spaces, we introduced *ordered relational structures*.

Definition 1 *An ordered Σ-relational structure is a partially ordered set endowed with maps and relations*

$$(X, \leq, \{H_X\}_{H \in Lh \cup La}, \{R_X\}_{R \in Jh \cup Ja}, \{Q_X\}_{Q \in Mh \cup Ma})$$

where for every $H \in Lh$, $H_X : X \to X$ is order-preserving, for every $K \in La$, $K_X : X \to X$ is order-reversing, for every $R \in Jh \cup Ja$ with arity n, $R_X \subseteq X^{n+1}$ is an increasing relation, and for every $Q \in Mh \cup Ma$ with arity n, $Q_X \subseteq X^{n+1}$ is a decreasing relation.

The class of all ordered Σ-relational structures will be denoted ORS_Σ. For every ordered Σ-relational structure X, its set $\mathcal{O}(X)$ of order-filters can be endowed with a bounded lattice structure (where join is union, meet is intersection, $0 = \emptyset$ and $1 = X$) and with additional operators which can be defined in a canonical way starting from the maps and relations on X as shown in what follows.

Theorem 1 ([5, 24]) *Let $X \in \mathsf{ORS}_\Sigma$. The following hold.*

1. *Every order preserving (resp. order reversing) operation on X, $H \in Lh$ (resp. $K \in La$) induces a lattice morphism (resp. lattice antimorphism) $h_H : \mathcal{O}(X) \to \mathcal{O}(X)$, (resp. $k_K : \mathcal{O}(X) \to \mathcal{O}(X)$) defined for every $U \in \mathcal{O}(X)$ by $h_H(U) = H_X^{-1}(U)$, (resp. $k_K(U) = X \setminus K_X^{-1}(U)$).*

2. *Every increasing relation $R_X \subseteq X^{n+1}$, $R \in Jh$ (resp. $R'_X \subseteq X^{n+1}$, with $R' \in Ja$), induces a join hemimorphism $f_R : \mathcal{O}(X)^n \to \mathcal{O}(X)$ (resp. join hemiantimorphism $f_{R'} : \mathcal{O}(X)^n \to \mathcal{O}(X)$), defined for every $U_1, \ldots, U_n \in \mathcal{O}(X)$ by*

(Jh) $f_R(U_1, \ldots, U_n) = \{x \in X \mid \exists x_1 \in U_1, \ldots, x_n \in U_n : R_X(x_1, \ldots, x_n, x)\}$,

(Ja) $f_{R'}(U_1, \ldots, U_n) = \{x \in X \mid \exists x_1 \notin U_1, \ldots, x_n \notin U_n : R'_X(x_1, \ldots, x_n, x)\}$.

   Every decreasing relation $Q_X \subseteq X^{n+1}$, $Q \in Mh$ (resp. $Q'_X \subseteq X^{n+1}$, with $Q' \in Ma$), induces a meet hemimorphism $g_Q : \mathcal{O}(X)^n \to \mathcal{O}(X)$ (resp. meet hemiantimorphism $g_{Q'} : \mathcal{O}(X)^n \to \mathcal{O}(X)$), defined for every $U_1, \ldots, U_n \in \mathcal{O}(X)$ by

(Mh) $g_Q(U_1, \ldots, U_n) = \{x \in X \mid \forall x_1, \ldots, x_n (Q_X(x_1, \ldots, x_n, x) \to \exists i, x_i \in U_i)\}$,

(Ma) $g_{Q'}(U_1, \ldots, U_n) = \{x \in X \mid \forall x_1, \ldots, x_n (Q'_X(x_1, \ldots, x_n, x) \to \exists i, x_i \notin U_i)\}$.

3. *A Heyting algebra structure can be defined on $\mathcal{O}(X)$ by defining for every $U_1, U_2 \in \mathcal{O}(X)$*

$$U_1 \Rightarrow U_2 = \{x \in X \mid \forall y, x \leq y \text{ if } y \in U_1 \text{ then } y \in U_2\}$$

$$\neg U_1 = \{x \in X \mid \forall y, x \leq y \text{ implies } y \notin U_1\}$$

In what follows we consider Kripke-style models for logics $\mathcal{L}$ with logical connectives[1] in $\{\vee, \wedge\} \cup \Sigma$, whose formulae are built from propositional variables taken from a (countable) set Var, and satisfying the following condition:

(L) $\mathcal{L}$ is sound and complete w.r.t. a subvariety $\mathcal{V}_\mathcal{L}$ of DLO_Σ, such that the Priestley duality induces a dual equivalence between $\mathcal{V}_\mathcal{L}$ and a suitable class $\mathcal{V}_\mathcal{L}\mathsf{Sp}$ of relational Priestley spaces.

Definition 2 *$\mathcal{K}_\mathcal{L}$ is the subclass of ORS_Σ consisting of all relational structures that satisfy all the conditions not involving the topology which characterize the class $\mathcal{V}_\mathcal{L}\mathsf{Sp}$.*

The Kripke models for $\mathcal{L}$ we consider here are relational structures[2] $(X, \leq, \{R_X\}_{R\in\Sigma})$ in $\mathcal{K}_\mathcal{L}$, endowed with a hereditary meaning function $m : \mathsf{Var} \to \mathcal{O}(X)$.

Definition 3 *Let $X \in \mathsf{ORS}_\Sigma$, let $m : \mathsf{Var} \to \mathcal{O}(X)$ be a meaning function, $\overline{m} : \mathsf{Fma}(\mathcal{L}) \to \mathcal{O}(X)$ its unique extension to a morphism of algebras, let $x \in X$, and let $\phi \in \mathsf{Fma}(\mathcal{L})$. We define:*
(1) $X \overset{r}{\models}_{m,x} \phi$ iff $x \in \overline{m}(\phi)$;
(2) $X \overset{r}{\models}_{m} \phi$ iff $\overline{m}(\phi) = X$;
(3) $X \overset{r}{\models} \phi$ iff $X \overset{r}{\models}_{m} \phi$ for every $m : \mathsf{Var} \to \mathcal{O}(X)$.

Lemma 2 *For any $X \in \mathsf{ORS}_\Sigma$, $X \overset{r}{\models} \phi$ iff $\mathcal{O}(X) \overset{a}{\models} \phi = 1$.*

Theorem 3 *Let $\mathcal{L}$ be a logic that satisfies condition (L). If for every $X \in \mathcal{K}_\mathcal{L}$, $\mathcal{O}(X) \in \mathcal{V}_\mathcal{L}$, then $\mathcal{L}$ is sound and complete w.r.t. the class $\mathcal{K}_\mathcal{L}$. Assume that $\mathcal{V}_\mathcal{L} = HSP(\mathcal{A})$ and let $\mathcal{K}_\mathcal{A}$ be the class of all ordered relational structures satisfying all non-topological conditions corresponding to the Priestley duals of the elements of $\mathcal{A}$. If for every $X \in \mathcal{K}_\mathcal{A}$, $\mathcal{O}(X) \in \mathcal{V}_\mathcal{L}$, then $\mathcal{L}$ is sound and complete w.r.t. $\mathcal{K}_\mathcal{A}$.*

Proof: Without loss of generality we only prove the second part of the theorem. Let ϕ be a theorem of $\mathcal{L}$. Then $\mathcal{V}_\mathcal{L} \overset{a}{\models} \phi = 1$. Since, by assumption, for every $X \in \mathcal{K}_\mathcal{A}$, $\mathcal{O}(X) \in \mathcal{V}_\mathcal{L}$, it follows that $X \overset{r}{\models} \phi$ for every $X \in \mathcal{K}_\mathcal{A}$.

Conversely, assume that for every $X \in \mathcal{K}_\mathcal{A}$, $X \overset{r}{\models} \phi$. Let $A \in \mathcal{A}$ be an algebra, and $f : \mathsf{Var} \to A$ an arbitrary valuation. By the Priestley duality for $\mathcal{V}_\mathcal{L}$, there exists an isomorphism $\eta_A : A \simeq E(D(A))$. Let $\iota : E(D(A)) \to \mathcal{O}(D(A))$ be the inclusion, and let $m : \mathsf{Var} \to \mathcal{O}(D(A))$ be defined by $m = \iota \circ \eta_A \circ f$. It is easy to see that $\overline{m} = \iota \circ \eta_A \circ \overline{f}$. We know that $D(A) \in \mathcal{K}_\mathcal{A}$. Then $\overline{m}(\phi) = D(A)$, hence $\overline{f}(\phi) = 1$. Since $\mathcal{V}_\mathcal{L} = HSP(\mathcal{A})$ and $\mathcal{L}$ is sound and complete w.r.t. $\mathcal{V}_\mathcal{L}$ it follows that $\vdash_\mathcal{L} \phi$. □

Corollary 4 *If $\mathcal{V}_\mathcal{L}$ is finitely generated by the family $\{A_1, \ldots, A_n\}$ of finite algebras in $\mathcal{V}_\mathcal{L}$ then $\mathcal{L}$ is sound and complete w.r.t. $\mathcal{K} = \{D(A_1), \ldots, D(A_n)\}$.*

A similar idea occurs in the work of Jónsson and Tarski [10] on Boolean algebras with operators. We also analyzed other representation theorems, e.g. for orthomodular lattices, which can be used with similar results. A presentation of these results is beyond the scope of the present paper.

4 Automated theorem proving

Translation to clause form. Let $\mathcal{L}$ be a propositional logic with a set Var of propositional variables, and satisfying condition (L), and let $\mathcal{K}_\mathcal{L}$ be as in Definition 2. In what follows we assume that $\mathcal{L}$ is sound and complete w.r.t. $\mathcal{K}_\mathcal{L}$.

Lemma 5 *For every $(X, \leq, \{R_X\}_{R\in\Sigma}) \in \mathcal{K}_\mathcal{L}$ and every $m : \mathsf{Var} \to \mathcal{O}(X)$; for every $H \in Lh, K \in La, R \in Jh, Q \in Mh, R' \in Ja$ and $Q' \in Ma$ the following hold:*

$X \overset{r}{\models}_{m,x} p$	iff	$x \in m(p)$	if $p \in \mathsf{Var}$
$X \overset{r}{\models}_{m,x} \phi_1 \vee \phi_2$	iff	$X \overset{r}{\models}_{m,x} \phi_1$ or $X \overset{r}{\models}_{m,x} \phi_2$	
$X \overset{r}{\models}_{m,x} \phi_1 \wedge \phi_2$	iff	$X \overset{r}{\models}_{m,x} \phi_1$ and $X \overset{r}{\models}_{m,x} \phi_2$	
$X \overset{r}{\models}_{m,x} H(\phi)$	iff	$X \overset{r}{\models}_{m,H_X(x)} \phi$	
$X \overset{r}{\models}_{m,x} K(\phi)$	iff	$X \overset{r}{\not\models}_{m,K_X(x)} \phi$	
$X \overset{r}{\models}_{m,x} R(\phi_1, \ldots, \phi_n)$	iff	$\exists x_1, \ldots, x_n (R_X(x_1, \ldots, x_n, x),$ $X \overset{r}{\models}_{m,x_1} \phi_1, \ldots, X \overset{r}{\models}_{m,x_n} \phi_n)$	
$X \overset{r}{\models}_{m,x} Q(\phi_1, \ldots, \phi_n)$	iff	$\forall x_1, \ldots, x_n (Q_X(x_1, \ldots, x_n, x) \to$ $\exists i \in \{1, \ldots, n\}, X \overset{r}{\models}_{m,x_i} \phi_i)$	
$X \models_{m,x} R'(\phi_1, \ldots, \phi_n)$	iff	$\exists x_1, \ldots, x_n (R'_X(x_1, \ldots, x_n, x),$ $X \overset{r}{\not\models}_{m,x_1} \phi_1, \ldots, X \overset{r}{\not\models}_{m,x_n} \phi_n)$	
$X \overset{r}{\models}_{m,x} Q'(\phi_1, \ldots, \phi_n)$	iff	$\forall x_1, \ldots, x_n (Q'_X(x_1, \ldots, x_n, x) \to$ $\exists i \in \{1, \ldots, n\}, X \overset{r}{\not\models}_{m,x_i} \phi_i)$	

Proof: *(Sketch)* Follows from the definition of $\overset{r}{\models}$ and from the definitions of the operations on $\mathcal{O}(X)$ induced by operations or relations in Lh, La, Jh, Mh, Ja, Ma, as given in Theorem 1. □

If $\mathcal{K}_\mathcal{L}$ is an elementary class (i.e. it is the class of all ordered Σ-relational structures satisfying a set E of first-order formulae), Lemma 5 can be used in order to define a *structure-preserving* transformation method to clause form. This structure-preserving translation to clause form is inspired by the well-known method from classical logic due to Tseitin: for every subformula ψ of ϕ a new (unary) predicate symbol P_ψ is introduced. The fact that this renaming is correct is expressed by a set (Ren) of formulae.

[1] The connectives in Σ correspond to versions of non-classical negation, or are generalizations of operators from modal logic such as those modeling knowledge, belief, necessity, or possibility.

[2] Further considerations lead to relaxing the condition that $\leq$ is a partial order to $\leq$ being reflexive and transitive. Then $\mathcal{O}(X)$ is the family of all increasing subsets of X w.r.t. this relation.

Proposition 6 *Let $\phi \in \mathsf{Fma}(\mathcal{L})$. Then $\vdash_{\mathcal{L}} \phi$ iff the following conjunction of formulae is unsatisfiable (where the predicates P_ψ are indexed by subformulae ψ of ϕ):*

$$
\begin{cases}
\text{(Neg)} & \exists x \neg P_\phi(x) \\
\text{(Ren)} & \forall x[P_{\psi_1 \vee \psi_2}(x) \leftrightarrow P_{\psi_1}(x) \vee P_{\psi_2}(x)] \\
& \forall x[P_{\psi_1 \wedge \psi_2}(x) \leftrightarrow P_{\psi_1}(x) \wedge P_{\psi_2}(x)] \\
& \forall x[P_{H(\psi)}(x) \leftrightarrow P_\psi(H(x))] \\
& \forall x[P_{K(\psi)}(x) \leftrightarrow \neg P_\psi(K(x))] \\
& \forall x[P_{R(\psi_1,\ldots,\psi_n)}(x) \leftrightarrow \\
& \quad \exists x_1,\ldots,x_n(R(x_1,\ldots,x_n,x) \wedge \bigwedge_{i=1}^n P_{\psi_i}(x_i))] \\
& \forall x[P_{Q(\psi_1,\ldots,\psi_n)}(x) \leftrightarrow \\
& \quad \forall x_1,\ldots,x_n(Q(x_1,\ldots,x_n,x) \rightarrow \bigvee_{i=1}^n P_{\psi_i}(x_i))] \\
& \forall x[P_{R'(\psi_1,\ldots,\psi_n)}(x) \leftrightarrow \\
& \quad \exists x_1,\ldots,x_n(R'(x_1,\ldots,x_n,x) \wedge \bigwedge_{i=1}^n \neg P_{\psi_i}(x_i))] \\
& \forall x[P_{Q'(\psi_1,\ldots,\psi_n)}(x) \leftrightarrow \\
& \quad \forall x_1,\ldots,x_n(Q'(x_1,\ldots,x_n,x) \rightarrow \bigvee_{i=1}^n \neg P_{\psi_i}(x_i))] \\
\text{(Str)} & \forall x\ x \leq x \\
& \forall x,y,z\,[(x \leq y \wedge y \leq z) \rightarrow x \leq z] \\
& \forall x,y\,[(x \leq y \wedge P_\psi(x)) \rightarrow P_\psi(y)] \\
& E \ \text{(the set of first-order formulae that characterize } \mathcal{K}_{\mathcal{L}}\text{)}
\end{cases}
$$

Proof: *(Sketch)* It follows from the fact that for every Kripke-style model (X, m), where $X \in \mathcal{K}_{\mathcal{L}}$ and $m : \mathsf{Var} \rightarrow \mathcal{O}(X)$ such that $X \not\models_m \phi$ we can construct a first-order model of (Neg) $\cup$ (Ren) $\cup$ (Str), and vice versa. □

The set of formulae (Neg) $\cup$ (Ren) $\cup$ (Str) can be transformed to clause form in the usual way. Proposition 7 gives an estimate of the number of clauses generated from a given formula ϕ by using Proposition 6.

Proposition 7 *Let ϕ be a formula of $\mathcal{L}$ containing logical connectives in $\{\vee, \wedge\} \cup Lh \cup La \cup Jh \cup Mh \cup Ja \cup Ma$. Let l be the number of subformulae of ϕ, and let m be the highest among the arities of the operators occurring in ϕ. Then the number of clauses generated using the rules in Proposition 6 is at most $l(m+2)+s$, where s is the number of clauses induced by the formulae in* (Str) $\cup$ (Neg).

Proof: It follows from the fact that the formulae in (Ren) introduce, for every operation in $\{\vee, \wedge\}$, 3 clauses; for every operation in $Lh \cup La$ 2 clauses; and for every n-ary operation in $Jh \cup Mh \cup Ja \cup Ma$, $(n+2)$ clauses. □

Translation to clause form for finitely-valued logics. If $\mathcal{L}$ is a finitely-valued logic having as algebra of truth values a distributive lattice with operators the procedure described above can be further extended. The Priestley dual $D(A)$ of A can be seen as a finite Kripke-style frame with respect to which, by Corollary 4, $\mathcal{L}$ is sound and complete. Even a version of first-order finitely-valued logics with quantifiers $\forall, \exists$ (interpreted as generalized meets resp. joins in A), defined by following the ideas in [1], is tractable this way. The rules for translating to clause form quantified formulae essentially use the fact that every element in the lattice of truth values can be uniquely written as a join of join-irreducible elements [9, 21, 23]; this argument cannot be applied if the algebra of truth values is infinite. For details cf. [21, 23].

For finitely-valued logics the use of quantification over the possible worlds can be avoided: the existential and universal quantifier can be replaced with joins resp. meets over all the elements of $D(A)$, which is a finite set. The advantage is that (for propositional logics) in the clause generation process, we remain inside propositional logic. The disadvantage is that, in the presence of logical connectives in Σ of arity greater than 1, it may lead to long clause forms.

Theorem proving. The translation to clause form described in this section is a translation to classical first-order logic; thus general methods for automated theorem proving in classical logic can be applied to the resulting sets of clauses.

5 Examples

We now illustrate the construction of classes of Kripke-style models and the translation to clause form presented above by some examples: logics based on (finite) p-lattices, logics based on implicative algebras, logics based on Ockham algebras, relevant logics. Due to space limitations, only one example is presented in detail; for the others the main ideas (especially concerning the use of Priestley duality for constructing classes of Kripke models) are only sketched. Other examples, such as SHn-logics or $SHKn$-logics, are presented in detail in [21].

5.1 Logics based on p-lattices

A distributive p-lattice is an algebra $(A, \vee, \wedge, *, 0, 1)$ of type $(2, 2, 1, 0, 0)$ such that $(A, \vee, \wedge, 0, 1)$ is a distributive lattice with 0 and 1, and $*$ is an operation of pseudocomplementation, i.e. $a^* = \max\{b \in A \mid a \wedge b = 0\}$.

It is known that the distributive p-lattices form a variety B_ω, and that the subvarieties of B_ω form a chain $\mathsf{B}_{-1} \subset \mathsf{B_0} \subset \ldots \subset \mathsf{B_n} \subset \ldots \subset \mathsf{B}_\omega$, where B_{-1} is the trivial variety, $\mathsf{B_0}$ the variety of Boolean algebras, and $\mathsf{B_n}$ $(n \geq 1)$ is generated by $P_n = 2^n \oplus 1$ (the n-th power of the 2-element chain to which a new top element is adjoined).

Lemma 8 *If a logic $\mathcal{L}$ is sound and complete w.r.t. B_ω, it is sound and complete w.r.t. the class of all Kripke models $((X, \leq), m)$, where $\leq$ is a partial order[3] on X, m is a meaning function, and the satisfiability relation is defined as in Lemma 5 for $\phi_1 \vee \phi_2, \phi_1 \wedge \phi_2$, and, additionally, by:*

$$X \overset{r}{\models}_{m,x} \phi^* \ \text{ iff } \ \forall y(x \leq y \rightarrow X \overset{r}{\not\models}_{m,y} \phi).$$

Proof: Follows from the duality theorem for p-lattices or Heyting algebras (cf. [18]) and Theorem 3, and, in particular, from the fact that for every meaning function $m : \mathsf{Var} \rightarrow \mathcal{O}(X)$, and every $x \in X$, $x \in \overline{m}(\phi^*) = \neg(\overline{m}(\phi)) = \{z \mid \forall y, z \leq y \rightarrow y \notin \overline{m}(\phi)\}$ iff $\forall y(x \leq y \rightarrow y \notin \overline{m}(\phi))$. □

[3] One can in fact prove soundness and completeness w.r.t. the class of the Kripke frames $(X, \leq)$, where $\leq \subseteq X \times X$ is reflexive and transitive.

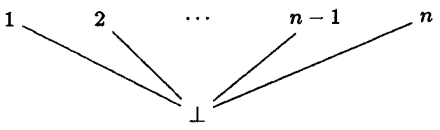


Figure 1. $V_n = D(P_n)$

For the translation to clause form, the semantics of the logical connectives $\vee, \wedge$ is used, as explained in Section 4. Additionally, for formulae starting with the connective $*$ the following formula has to be added to (Ren):

$$\forall x[P_{\phi^*}(x) \leftrightarrow (\forall y(x \leq y \rightarrow \neg P_\phi(y)))].$$

Logics based on finite distributive p-lattices. Concerning logics having P_n as an algebra of truth values, we make the following remarks about the use of the Priestley representation: the Priestley dual of $P_n = 2^n \oplus 1$ is the space V_n, represented in Figure 1. Thus, for every n, P_n is isomorphic to the lattice of order filters of V_n, where for every $U \subseteq \mathcal{O}(V_n)$, $U^* = X \setminus \downarrow U$. The finiteness of V_n leads to the following simplified rule for the pseudocomplementation $*$, without quantification over the elements of V_n:

$$P_{\phi^*}(\bot) \leftrightarrow \neg P_\phi(\bot) \wedge \neg P_\phi(1) \wedge \ldots \wedge \neg P_\phi(n)$$

$$P_{\phi^*}(i) \leftrightarrow \neg P_\phi(i) \text{ (for every } i \in \{1, \ldots, n\}).$$

Proposition 9 *The number of clauses generated from a formula ϕ in a logic based on B_ω is $\mathcal{O}(l)$, where l is the length of ϕ. In a logic based on P_n, the number of clauses generated from a formula ϕ is $\mathcal{O}(nl)$ and all clauses are ground.*

Proof: Since the formulae for renaming introduce at most 3 new clauses, in the case of logics based on B_ω, the number of clauses generated by the above procedure is $\mathcal{O}(l)$. In the case of logics based on P_n, due to the particular form of V_n, for every $x \in V_n \setminus \{\bot\}$ there is exactly one $y \in V_n$ with $y \geq x$. For $x = \bot$ there are exactly $n+1$ such elements. Thus, in this case the number of clauses generated from a formula ϕ is $1+3(k_1+k_2)(n+1)+k_3((n+2)+2n)$, where k_1, k_2, k_3 are the numbers of occurrences of $\vee, \wedge$, resp. $*$ in ϕ. It is easy to see that all these clauses are ground. □

Proposition 10 *Validity of propositional formulae in logics based on B_ω and P_n is decidable.*

Proof: If we analyze the form of the clauses in (Str) ∪ (Neg) ∪ (Ren) for a formula in a logic based on B_ω, we notice that we are in the situation studied in [4]. Hence, the validity of formulae in logics based on B_ω is decidable by ordered chaining with selection. The propositional logics based on P_n for some n are of course decidable. □

For first-order logics based on P_n one can prove the decidability of certain fragments by using classical decidability results from first-order logic. For further details cf. [22].

5.2 Other classes of logics

Intuitionistic logics. By the duality theorem for Heyting algebras (cf. e.g. [5]) and Theorem 3 it follows that intuitionistic logics are sound and complete with respect to the class of all Kripke models $(X, \leq)$, where $\leq$ is a partial order on X, a well-known result.

Logics based on Ockham algebras. Ockham algebras are distributive lattices with 0, 1, and one unary lattice antimorphism $\sim$. They are models for logics endowed with a negation operator that satisfies the De Morgan laws. A Priestley duality for Ockham algebras is given in [25]. The corresponding Priestley spaces are endowed with a unary, continuous, order-reversing map g. By Theorem 1 and Theorem 3, the logics which are sound and complete with respect to the variety of Ockham algebras have as Kripke-style models ordered spaces $(X, \leq, g)$ endowed with an order-reversing map (the condition that $\leq$ is a partial order can be relaxed to the condition that $\leq$ is reflexive and transitive).

Łukasiewicz logics of order n. We showed that, by using the Priestley-style duality for implicative algebras due to Martínez [12], in the case of Łukasiewicz algebras of order n we obtain a translation to clause form similar to the translation to mixed integer programming given in [6].

Relevance logics. Urquhart [26] gave an algebraic semantics for *relevance logics* in terms of *relevant algebras* (a class of distributive lattices with operators) and established a Priestley duality for this type of algebras. We showed that for every space X obeying all those properties of the Priestley duals of relevant algebras which do not involve topological notions, $\mathcal{O}(X)$ (with operations appropriately defined) is a relevant algebra. Thus, by Theorem 3, such spaces provide a family of Kripke-style models for relevant logics, which coincides with the Kripke models for a version of relevance logics studied by Routley and Meyer (cf. [19]). Theorem 3 shows that soundness and completeness with respect to this class of Kripke-style models can be proved by completely algebraical means, by using the soundness and completeness with respect to the class of relevant algebras.

6 Related Work

The work reported here was inspired by the papers of Baaz and Fermüller [1] and Hähnle [7] on resolution-based theorem proving in finite-valued quantificational first-order logics. It extends the ideas in [21] where we gave a method for translation to a signed clause form and signed resolution methods for (first-order) finitely-valued logics having a finite DLO A as algebra of truth values. The signed literals used in [21, 23] were of the form $\boxed{\alpha} L^t$ and $\boxed{\alpha} L^f$, where $\alpha \in D(A)$. Our method extended existing methods for automated theorem proving in *regular logics* [7] (there, the literals were of the form $\boxed{\geq i} L$ and resp. $\boxed{< i} L$, for

some truth value i) to cases when the set of truth values is not linearly ordered. In this paper we replaced the notation above by $L(\alpha)$ resp. $\neg L(\alpha)$ (the translation is one to classical logic [22]), and showed that, using additional theoretical considerations, the ideas we had for finitely-valued logics can be extended to more general propositional logics.

Among existing methods exploiting the structure of the algebra of truth values, we mention [16, 13, 15], who gave theorem proving systems for m-valued Post logics and for algorithmic logics, Salzer [20] who studied operators and distribution quantifiers in finitely-valued logics based on semi-lattices, and Hähnle [9] who derived tableau-style axiomatizations of distribution quantifiers by using Birkhoff's representation theorem for finite distributive lattices. Methods for automated proving in paraconsistent or annotated logics (cf. e.g. [11]) also have similarities with our approach in the case of finitely-valued logics. Known methods from modal logic seem to occur as particular cases of general concepts such as those considered here (see also [5], [14]).

7 Conclusions

In this paper we showed that the Priestley duality for distributive lattices with operators provides a theoretical tool for clarifying the link between algebraic and Kripke-style models for certain classes of non-classical logics. Based on these results, we gave a method for translation to clause form and automated theorem proving for some of these classes of non-classical logics. We expect that this will enable us to use recent results in the algebraic investigation of various classes of distributive lattices with operators for obtaining Kripke models and automated theorem proving procedures for logics with interesting practical applications. Until now we only studied the existence of decision procedures in certain cases; in the future we intend to continue our work in this direction.

Acknowledgements. We gratefully acknowledge the support by ESPRIT BRP 6471 at RISC-Linz, by a postdoctoral fellowship at MPII, and by COST Action 15 MVL. We thank M. Baaz, H. Ganzinger, R. Hähnle, L. Iturrioz, H. de Nivelle, H.J. Ohlbach, E. Orłowska, J. Pfalzgraf, H. Priestley, and K. Stokkermans for stimulating discussions. We also thank the referees for their helpful comments.

References

[1] M. Baaz and C. Fermüller. Resolution-based theorem proving for many-valued logics. *J. Symb. Comp.*, 19:353–391, 1995.

[2] S. Burris and H. Sankappanavar. *A Course in Universal Algebra.* Graduate Texts in Mathematics. Springer, 1981.

[3] B. Davey and H. Priestley. *Introduction to Lattices and Order.* Cambridge University Press, 1990.

[4] H. Ganzinger, C. Meyer, U. Hustadt, and R. Schmitt. A resolution-based decision procedure for extensions of K4. Advances in Modal Logic, 1998.

[5] R. Goldblatt. Varieties of complex algebras. *Ann. Pure Appl. Logic*, 44(3):153–301, 1989.

[6] R. Hähnle. Many-valued logic and mixed integer programming. *Ann. Math. and AI*, 12:231–263, 1994.

[7] R. Hähnle. Short conjunctive normal forms in finitely valued logics. *J. Logic and Comp.*, 4(6):905–927, 1994.

[8] R. Hähnle. Exploiting data dependencies in many-valued logics. *J. Appl. Non-Classical Logics*, 6(1):49–69, 1996.

[9] R. Hähnle. Commodious axiomatization of quantifiers in multiple-valued logic. *Studia Logica*, 61(1):101–121, 1998.

[10] B. Jónsson and A. Tarski. Boolean algebras with operators, I, II. *American J. of Mathematics* 73:891–939, 1951 and 74:127–162, 1952.

[11] M. Kifer and M. Lozinskii. A logic for reasoning with inconsistency. *J. Automated Reasoning*, 9:179–215, 1992.

[12] N. Martínez. A topological duality for some ordered lattice ordered algebraic structures including l-groups. *Algebra Universalis*, 31:516–541, 1994.

[13] C. Morgan. A resolution principle for a class of many-valued logics. *Logique et Analyse*, 19(74-76):311–339, 1976.

[14] H. Ohlbach. Translation methods for non-classical logics: An overview. *Bull. of the IGPL*, 1(1):69–89, 1993.

[15] E. Orłowska. Resolution systems and their applications I, II. *Fundamenta Informaticae* 3:235–268, 1979; 3:333–362, 1980.

[16] E. Orłowska. The resolution principle for ω^+-valued logic. *Fundamenta Informaticae*, 2:1–15, 1978.

[17] H. Priestley. Representation of distributive lattices by means of ordered Stone spaces. *Bull. London Math. Soc.*, 2:186–190, 1970.

[18] H. Priestley. Ordered sets and duality for distributive lattices. *Ann. Discr. Math.*, 23:39–60, 1984.

[19] R. Routley, R. Meyer, V. Plumwood, and R. Brady. *Relevant Logics and Their Rivals 1.* Ridgeview Publ. Comp., 1982.

[20] G. Salzer. Optimal axiomatizations for multiple-valued operators and quantifiers based on semilattices. In M. McRobbie and J. Slaney, editors, *Proc. 13th CADE*, LNCS 1104, pages 688–702. Springer, 1996.

[21] V. Sofronie-Stokkermans. *Fibered Structures and Applications to Automated Theorem Proving in Certain Classes of Finitely-Valued Logics and to Modeling Interacting Systems.* PhD thesis, RISC-Linz, J.Kepler Univ. Linz, 1997.

[22] V. Sofronie-Stokkermans. On translation of finitely-valued logics to classical first-order logic. In H. Prade, editor, *Proceedings of ECAI 98*, pages 410–411. John Wiley, 1998.

[23] V. Sofronie-Stokkermans. Automated theorem proving by resolution for finitely-valued logics based on distributive lattices with operators. Submitted, 1999.

[24] V. Sofronie-Stokkermans. Duality and canonical extensions of bounded distributive lattices with operators, and applications to the semantics of non-classical logics I, II. *Studia Logica. To appear*, 1999.

[25] A. Urquhart. Distributive lattices with a dual homomorphic operation. *Studia Logica*, 38(2):201–209, 1979.

[26] A. Urquhart. Duality for algebras of relevant logics. *Studia Logica*, 56(1,2):263–276, 1996.

Transformations between Signed and Classical Clause Logic*

Bernhard Beckert Reiner Hähnle
University of Karlsruhe
Institute for Logic, Complexity
and Deduction Systems
D-76128 Karlsruhe, Germany
{beckert,reiner}@ira.uka.de

Felip Manyà†
Universitat de Lleida
Pl. Víctor Siurana 1
E-25003 Lleida, Spain
felip@eup.udl.es

Abstract

In the last years two automated reasoning techniques for clause normal form arose in which the use of labels are prominently featured: signed logic *and* annotated logic programming, *which can be embedded into the first. The underlying basic idea is to generalise the classical notion of a literal by adorning an atomic formula with a sign or label which in general consists of a possibly ordered set of truth values. In this paper we relate signed logic and classical logic more closely than before by defining two new transformations between them. As a byproduct we obtain a number of new complexity results and proof procedures for signed logics.*

1 Introduction

In the last years two automated reasoning techniques for clause normal form arose in which the use of labels are prominently featured: from generic treatments of many-valued logic, so-called *signed logic* emerged (see, for example, [6, 7, 3, 4, 14, 15]) while *annotated logic programming* (see, for example, [12, 8, 9]) was motivated by attempts to deal with inconsistency in deductive databases. Both approaches are closely connected to each other [13, 10] and to constraint logic programming [11]. In fact, annotated logic can be embedded into signed logic [13].

In each case the underlying basic idea is to generalise the classical notion of a literal by adorning an atomic formula with a sign or label, which in general consists of a finite set of (truth) values. Whenever the values appearing in the signs are partially ordered, polarities can be assigned to signed literals in a natural way which gives rise to generalised notions of a Horn set. It turns out that many problems can be represented more succinctly using formulae over signed literals whose proof procedures and complexities are often (but not always) similar as in classical logic.

In the present paper we relate signed logic and classical logic more closely than it has been done before. This is done by defining two new transformations between them. After formal definition of some basic notions in the next section we start in Section 3 with transforming arbitrary classical formulae in conjunctive normal form (CNF) into signed CNF formulae *with at most two literals per clause*. This provides an alternative proof of NP-hardness of signed 2-SAT (first proved by [14]) and creates the possibility to compare classical and signed deduction procedures experimentally. In Section 4.1 we take the reverse direction and reduce signed Horn formulae based on certain partial orders to classical logic. In the case of lattice orders this yields the new result that generalised Horn problems turn out to have still polynomial complexity with respect to formula size and number of truth values (Section 4.2). We can also extract an efficient decision procedure based on generalised unit resolution (Section 4.3). A major advantage of our reduction to classical logic is that it scales up: we demonstrate this by sketching generalisations to infinite orders in Section 4.4 and to partial orders that are not lattices in Section 4.5.

Due to space limitations, most proofs had to be omitted. A full version of this paper is available from the authors on request.

2 Basic Definitions

2.1 Syntax of Signed Logic

Definition 1. A *truth value set* is a non-empty, finite set $N = \{i_1, i_2, \ldots, i_n\}$. The cardinality of N is denoted $|N|$.

*This work has been partially supported by EC COST Action 15 *Many-valued Logics for Computer Science Applications*.

†Partially supported by the project TIC96-1038-C04-03 funded by the CICYT. This work was carried out during a visit to the University of Karlsruhe with a postdoctoral fellowship of the "Comissionat per a Universitats i Recerca" (1997BEAI400138).

0-7695-0161-3/99 $10.00 © 1999 IEEE

Definition 2. Let Σ be a *propositional signature*, i.e., a denumerable set of propositional variables. We define the *set of atomic signed formulae* (or signed atoms for short) as:

$$\{S : p_i \mid S \subseteq N, p_i \in \Sigma\} \ .$$

Definition 3. Given a signed atom $S : p$, then S is said to be its *sign*. Let $\geq$ be a partial order on the truth value set N, let $\uparrow i$ denote the set $\{j \in N \mid j \geq i\}$ and let $\downarrow i$ denote the set $\{j \in N \mid j \leq i\}$. If a sign S is equal to either $\uparrow i$ or $\downarrow i$, for some $i \in N$, then it is called a *regular sign*. A signed atom with a regular sign is called a *regular atom*.

Definition 4. A *signed clause* C is an expression of the form

$$S_1 : p_1, \ldots, S_k : p_k \ \rightarrow \ S'_1 : q_1, \ldots, S'_l : q_l \ ,$$

where $S_1 : p_1, \ldots, S_k : p_k$ and $S'_1 : q_1, \ldots, S'_l : q_l$ are signed atoms and $k, l \geq 0$.

We say the signed atoms $S_1 : p_1, \ldots, S_k : p_k$ occur with *negative polarity* in C, the signed atoms $S'_1 : q_1, \ldots, S'_l : q_l$ occur with *positive polarity*. The expression on the left of $\rightarrow$ is called the *body* of the clause and the expression on the right is called the *head*. A *signed formula* in conjunctive normal form (CNF) is a finite set of signed clauses.

A signed clause is called *regular* if $(N, \geq)$ is a lattice and it only contains regular atoms with signs of the form $\uparrow i$.[1] A signed CNF formula is called *regular* if it only contains regular clauses. A regular clause containing at most one atom with positive polarity is a *regular Horn clause*. A regular CNF formula consisting solely of regular Horn clauses is a *regular Horn formula*.

In signed clauses $k = 0$ and $l = 0$ are allowed; thus, for signed atoms p, q, both $p, q \ \rightarrow \ \langle\rangle$ and $\langle\rangle \ \rightarrow \ p, q$ are signed clauses, and we represent them by $p, q \ \rightarrow$ and $\rightarrow \ p, q$. When $k = 0$ and $l = 0$ we have the signed empty clause, denoted by $\square$.

Definition 5. The *length* of a *signed atom* $S : p$, denoted by $|S : p|$, is $|S|+1$, where $|S|$ denotes the cardinality of S. The *length* of a *signed clause* C, denoted by $|C|$, is the sum of the lengths of the signed atoms occurring in C. The *length* of a *signed CNF formula* Γ, denoted by $|\Gamma|$, is the sum of the lengths of the clauses in Γ.

2.2 Semantics of Signed Logic

Definition 6. An *interpretation* is a mapping that assigns to every propositional variable of the signature Σ a truth value of N. An interpretation I *satisfies* a *signed atom* $S : p$, in symbols $I \models S : p$, iff $I(p) \in S$; I *satisfies* a *signed clause* $C = S_1 : p_1, \ldots, S_k : p_k \ \rightarrow \ S'_1 : q_1, \ldots, S'_l : q_l$, denoted $I \models C$, iff the following holds: If I satisfies all of $S_1 : p_1, \ldots, S_k : p_k$ then I must also satisfy at least one of $S'_1 : q_1, \ldots, S'_l : q_l$. A signed CNF formula Γ is *satisfiable* iff there exists an interpretation I that satisfies all the signed clauses in Γ. We say then that I is a model of Γ and we write $I \models \Gamma$. A signed CNF formula that is not satisfiable is *unsatisfiable*. The signed empty clause is always unsatisfiable and the signed empty CNF formula is always satisfiable.

In particular, I satisfies $\rightarrow \ S'_1 : q_1, \ldots, S'_l : q_l$ iff it satisfies at least one of $S'_1 : q_1, \ldots, S'_l : q_l$; similarly, I satisfies $S_1 : p_1, \ldots, S_k : p_k \ \rightarrow$ iff it does *not* satisfy at least one of $S_1 : p_1, \ldots, S_k : p_k$.

Observe that if we take $N = \{true, false\}$, assuming $true > false$, and consider only regular atoms of the form $\uparrow true : p$, then we obtain the logic of classical conjunctive normal form: $\uparrow true : p$ is equivalent to the classical atom p if it occurs with positive polarity, and to the negated classical atom $\neg p$ if it occurs with negative polarity. So, the classical clause $p_1, \ldots, p_k \ \rightarrow \ q_1, \ldots, q_l$ is equivalent to the regular clause

$$\uparrow true : p_1, \ldots, \uparrow true : p_k \ \rightarrow \\ \uparrow true : q_1, \ldots, \uparrow true : q_l \ .$$

In the following, when we refer to classical clauses we use the former notation.

In classical propositional logic, clauses are sometimes defined as a finite disjunction of literals (i.e., signed atoms or negated signed atoms). It is easy to see from the previous definitions that $p_1, \ldots, p_k \ \rightarrow \ q_1, \ldots, q_l$ is logically equivalent to $\neg p_1 \vee \cdots \vee \neg p_k \vee q_1 \vee \cdots \vee q_l$. So, classical atoms occurring with negative polarity are implicitly negated. In our definition of signed clauses, signed atoms occurring with negative polarity are implicitly negated as well in the sense that a signed atom $S : p$ with negative polarity is satisfied by an interpretation I iff $I \not\models S : p$. Nevertheless, we do not define regular clauses as a disjunction of regular atoms with arbitrary regular signs since, as we assume N to be *partially* ordered, an occurrence of $\uparrow i : p$ with negative polarity is not, in general, logically equivalent to $\downarrow j : p$ for any $j \in N$ and can therefore not be represented by a positive occurrence of some literal $\downarrow j : p$.

2.3 Satisfiability Problems

The propositional satisfiability problem, briefly called SAT, is the problem to determine whether a classical CNF formula is satisfiable, and is known for being the original NP-complete problem [1]. However, there exist linear-time algorithms for solving the SAT problem when we consider

[1] Regular clauses could also be defined containing only signs of the form $\downarrow i$ instead of signs of the form $\uparrow i$. The results of this paper are also valid for regular clauses defined that way.

Horn formulae (Horn SAT) [2] or CNF formulae with only two literals per clause (2-SAT) [5]. When a CNF formula admits three literals per clause (3-SAT), it is again an NP-complete problem.

In the last years, some results about the complexity of the satisfiability problems for different versions of signed CNF formulae have been published. These problems have the truth value set N (resp. $(N, \geq)$) as a second input parameter (besides the formula Γ to be tested for satisfiability). Thus, *signed SAT* is the problem of deciding for an arbitrary formula Γ over an arbitrary thruth value set N, whether there is an interpretation over N satisfying Γ. We also consider decision problems where N is not an input parameter but fixed, which we denote by attaching the fixed truth value set N as an index to the name of the decision problem. For example, given a fixed truth value set N, *signed SAT*$_N$ is the problem of deciding for an arbitrary formula Γ over N whether there is an interpretation over N satisfying Γ.

The classical SAT problem is trivially reducible to signed SAT$_{\{true,false\}}$; therefore, the latter and the more general problem signed SAT are NP-complete. Signed 2-SAT$_N$ is known to be NP-complete for $|N| \geq 3$ [14] (thus, the general problem signed 2-SAT is NP-complete). But, the restriction of signed 2-SAT to the case where all signs are singletons, called monosigned 2-SAT, is polynomially solvable [14]. Concerning regular CNF formulae, it is known that regular Horn SAT [7, 3] and regular 2-SAT [14] are both polynomially solvable in case the partial order defined over the set of truth values is total.

3 Transforming Classical SAT into Signed 2-SAT

3.1 The Transformation

In this section, a mapping $'$ is defined that transforms classical CNF formulae into signed (not necessarily regular) 2-CNF formulae, in other words, a reduction of classical SAT to signed 2-SAT; the transformation is shown to be computable in polynomial time. We define $'$ as follows: Let Γ be a classical CNF formula with clauses $C_1, \ldots, C_r$ ($r \geq 1$) over a signature Σ. Assume that $p_1, \ldots, p_s$ ($s \geq 1$) are the propositional variables occurring in Γ; thus, the clauses in Γ are of the form[2]

$$C_m = p_{i_{m,1}}, \ldots, p_{i_{m,k_m}} \rightarrow p_{j_{m,1}}, \ldots, p_{j_{m,l_m}} \ .$$

We associate with Γ a signed 2-CNF formula Γ' over the truth value set $N'_\Gamma = \{p_1^-, \ldots, p_s^-, p_1^+, \ldots, p_s^+\}$ and signature $\Sigma' = \{p'_1, \ldots, p'_r\}$, i.e., the truth values are the classical atoms annotated with the two possible polarities $-$ and $+$, and for each clause C_m in Γ there is a propositional variable p'_m in Σ'. The idea is that p'_m has the truth value p_i^+ or p_i^- in a (non-classical) interpretation I' if the classical atom p_i is the one that makes the clause C_m true in the corresponding classical interpretation I. That is, $I'(p'_m) = p_i^-$ if p_i is false in I and occurs with negative polarity in C_m, and $I'(p'_m) = p_i^+$ if p_i is true in I and occurs with positive polarity in C_m. An atom can only have a single truth value whereas a clause C_m can be "made true" by more than one of its literals, in which case an arbitrary one may be chosen to be the truth value of p'_m. For each clause

$$C_m \ = \ p_{i_{m,1}}, \ldots, p_{i_{m,k_m}} \rightarrow p_{j_{m,1}}, \ldots, p_{j_{m,l_m}}$$

in Γ there is a unit clause

$$C'_m \ = \ \rightarrow \{p^-_{i_{m,1}}, \ldots, p^-_{i_{m,k_m}}, p^+_{j_{m,1}}, \ldots, p^+_{j_{m,l_m}}\} : p'_m$$

in Γ'. The signed atom in C'_m represents the fact that C_m (the m-th clause of Γ) is made true. Thus Γ' represents only satisfying truth assignments of Γ.

This is, of course, not enough. We must ensure that Γ' in fact represents solely such truth assignments for atoms in Γ which are consistent or, in usual terminology, which are well-defined interpretations. For this purpose, Γ' contains for all (classical) clauses C_m and C_n in Γ. resp., for all propositional variables p'_m and p'_n in Σ' ($1 \leq m, n \leq r$) and for all atoms, resp., truth values p_i ($1 \leq i \leq s$) additional clauses

$$D'_{mni} \ = \ \{p_i^+\} : p'_m \rightarrow (S'_n \setminus \{p_i^-\}) : p'_n$$

where

$$S'_n = \{p^-_{i_{n,1}}, \ldots, p^-_{i_{n,k_n}}, p^+_{j_{n,1}}, \ldots, p^+_{j_{n,l_n}}\} \ .$$

The signed clauses D'_{mni} express that if an atom is used with positive polarity to "make true" some clause C_m of Γ, then it cannot be used with negative polarity to "make true" any other clause of Γ.

The clause D'_{mni} may be omitted from Γ' if p_i does not occur with positive polarity in C_m or does not occur with negative polarity in C_n. Instead of the clauses D'_{mni}, the clauses

$$E'_{mni} \ = \ \{p_i^-\} : p'_m \rightarrow (S'_n \setminus \{p_i^+\}) : p'_n$$

can be used. The proof of Theorem 7 shows that it is indeed sufficient to either use only the clauses D'_{mni} or only the clauses E'_{mni}.

Example 1. Consider the classical CNF formula Γ consisting of the clauses

$$\begin{array}{ll} (C_1) & p \rightarrow q \\ (C_2) & q \rightarrow p \\ (C_3) & \rightarrow p, q \end{array}$$

[2] Recall from Section 2 that the atoms in Γ really are signed atoms of the form $\uparrow true : p$, but the signs are not shown in representations of classical CNF formulae.

The only model I of Γ is defined by $I(p) = I(q) = \mathit{true}$. The result of applying the mapping $'$ to Γ is a signed 2-CNF formula Γ' over the signature $\Sigma' = \{p'_1, p'_2, p'_3\}$ and using the truth value set $N'_\Gamma = \{p^-, q^-, p^+, q^+\}$; Γ' consists of the clauses

$$\begin{array}{ll}
(C'_1) & \rightarrow \{p^-, q^+\} : p'_1 \\
(C'_2) & \rightarrow \{q^-, p^+\} : p'_2 \\
(C'_3) & \rightarrow \{p^+, q^+\} : p'_3 \\
(D'_{211}) & \{p^+\} : p'_2 \rightarrow \{q^+\} : p'_1 \\
(D'_{311}) & \{p^+\} : p'_3 \rightarrow \{q^+\} : p'_1 \\
(D'_{122}) & \{q^+\} : p'_1 \rightarrow \{p^+\} : p'_2 \\
(D'_{322}) & \{q^+\} : p'_3 \rightarrow \{p^+\} : p'_2
\end{array}$$

In (non-classical) interpretations I' satisfying Γ', the truth value of p'_1 is q^+, and the truth value of p'_2 is p^+. The truth value of p'_3 can be either p^+ or q^+, according to the fact that both atoms in the clause C_3 are satisfied by the classical interpretation I.

3.2 Results

The following theorem states the correctness of the transformation $'$:

Theorem 7. *A classical CNF formula Γ is satisfiable if and only if there is an interpretation over N'_Γ satisfying Γ'.*

Proof sketch. Only-if-part: Assume that the classical interpretation I satisfies Γ. Define the interpretation I' over N'_Γ as follows: In each clause $C_m \in \Gamma$ there has to be an atom p such that (1) $I(p) = \mathit{true}$ and p occurs positively in C_m or (2) $I(p) = \mathit{false}$ and p occurs negatively in C_m, because otherwise C_m were not satisfied by I. If there is more than one such atom p in C_m, then choose an arbitrary one. If (1) holds for p, then define $I'(p'_m) = p^+$, otherwise (i.e., if (2) holds for p) define $I'(p'_m) = p^-$. It is easy to show that I' satisfies the clauses C'_m and D'_{mni} for all $1 \leq m, n \leq r$ and $1 \leq i \leq s$.

If-part: Assume that the interpretation I' satisfies Γ'. We define the classical interpretation I for all atoms $p \in \Sigma$ as follows: If, for any $1 \leq m \leq r$, there is an atom p'_m such that $I'(p'_m) = p^+$, then let $I(p) = \mathit{true}$; let $I(p) = \mathit{false}$ otherwise. It is easy to show that I satisfies all clauses C_m in Γ. □

The size of Γ' is easily seen to be

$$\sum_m \underbrace{(k_m + l_m + 1)}_{=|C'_m|} + \sum_{m,n,i} \underbrace{(k_n + l_n + 2)}_{=|D'_{mni}|} ,$$

which is less than or equal to $|\Gamma| + r + 2r^2 s$ where r is the number of clauses in Γ, and s is the number of different atoms occurring in Γ. As $r, s < |\Gamma|$, this places $|\Gamma'|$ in $\mathcal{O}(|\Gamma|^3)$. Obviously, Γ' can be constructed in time which is linear in its own size and, thus, the time complexity of its construction is in $\mathcal{O}(|\Gamma|^3)$.

Theorem 8. *The transformation $'$ is computable in cubic time.*

In [14], NP-hardness of signed 2-SAT was proven with a poly-time reduction from 3-colourability of graphs to signed 2-SAT$_N$ with $|N| = 3$. As classical SAT is NP-complete, NP-hardness of signed 2-SAT (with N as an input parameter) follows as well as a corollary from Theorem 8.

Corollary 9. *Signed 2-SAT is NP-complete.*

An additional benefit of the transformation $'$ is that it makes it possible to compare classical decision procedures with specific procedures for signed CNF.

4 Transforming Regular Horn SAT into Classical Horn SAT

4.1 The Transformation

In this section, we define a mapping * that transforms lattice-ordered regular Horn formulae into classical Horn formulae; and we prove that it is linear in the size of the transformed formula and quadratic in the size of the truth-value lattice.

We assume in the following that the formula to be transformed does not contain a signed atom of the form $\uparrow \bot : p$, where $\bot$ is the bottom element of the truth value lattice. This is not a real restriction, as such atoms are true in all interpretations; they can be removed from a formula in linear time preserving satisfiability as follows: (1) if a clause contains a negative occurrence of $\uparrow \bot : p$, then remove that occurrence from the clause; (2) if a clause contains a positive occurrence of $\uparrow \bot : p$, then remove the whole clause from the formula.

The mapping * is defined as follows: Let Γ be a regular Horn formula over the truth-value lattice $(N, \geq)$ not containing the sign $\uparrow \bot$. Let $C_1, \ldots, C_r$ be the clauses in Γ ($r \geq 1$), let $p_1, \ldots, p_s \in \Sigma$ be the propositional variables occurring in Γ ($s \geq 1$).

We associate with Γ a classical Horn formula Γ^* over the signature

$$\Sigma^* = \{\uparrow i : p \mid i \in N, p \in \Sigma\} ,$$

that is signed atoms—including their signs—are used as propositional variables. A transformation based on the same principle is described in [16]; it allows to transform formulae from certain finite-valued logics whose truth value lattice is distributive into classical CNF formulae.

For each $1 \leq m \leq r$ the classical Horn formula Γ^* contains the clauses $C^*_m = C_m$ from Γ, which in Γ^* are regarded as classical clauses over Σ^*. In addition, for all truth values $i, j \in N$ and all propositional variables p_k occurring in Γ ($1 \leq k \leq s$), Γ^* contains

1. the clause

$$D^*_{ijk} \;=\; \uparrow i : p_k \rightarrow \uparrow j : p_k \;,$$

provided that (a) $i > j$ and (b) there is no $j' \in N$ such that $i > j' > j$,

2. the clause

$$E^*_{ijk} \;=\; \uparrow i : p_k, \uparrow j : p_k \rightarrow \uparrow (i \sqcup j) : p_k \;,$$

if neither $i \geq j$ nor $j \geq i$, where $i \sqcup j$ is the supremum of i and j in the truth value lattice.

As Γ^* contains the clauses from Γ, the classical interpretations satisfying Γ^* satisfy Γ as well. The additional clauses D^*_{ijk} and E^*_{ijk} ensure that such a classical interpretation I^* over the signature Σ^* corresponds to a well defined interpretation I over the signature Σ.

The clauses D^*_{ijk} represent the fact that, if I satisfies $\uparrow i : p_k$, i.e., $I(p_k) \geq i$ and $i > j$, then $I(p_k) \geq j$ and I satisfies $\uparrow j : p_k$ as well.

The clauses E^*_{ijk}, on the other hand, represent the fact that, if I satisfies both $\uparrow i : p_k$ and $\uparrow i : p_k$, i.e., $I(p_k) \geq i$ and $I(p_k) \geq j$, then $I(p_k) \geq i \sqcup j$ and, thus, $I \models \uparrow (i \sqcup j) : p_k$.

The precondition (a) $i > j$ for the inclusion of the clauses D^*_{ijk} in Γ^* is necessary for the correctness of the transformation; in case *not* $i > j$, the clauses D^*_{ijk} are (in general) not satisfied by arbitrary interpretations. Contrary to that, the precondition (b) for the inclusion of the D^*_{ijk} and the precondition for the inclusion of the clauses E^*_{ijk} is only needed to avoid redundancies.

The following lemma shows that clauses D^*_{ijk} for values of i, j violating precondition (b) are redundant. They are true in all interpretations satisfying Γ^*. Therefore, their inclusion would not impose any further restriction on the models of Γ^*.

Lemma 10. *Let Γ be a regular Horn formula over a signature Σ, let I^* be a classical interpretation satisfying Γ^*, let $p \in \Sigma$, and let j, j' be truth values in N such that*

$$j \geq j'$$

and

$$I^*(\uparrow j : p) = \mathit{true} \;;$$

then

$$I^*(\uparrow j' : p) = \mathit{true} \;.$$

The clauses E^*_{ijk} are tautological (and redundant), whenever $i \geq j$ or $j \geq i$; in particular, they are not needed if the ordering $\geq$ on the truth value set N is total.

According to the following lemma, it is not necessary to include in Γ^* clauses of the form

$$\uparrow i_1 : p, \ldots, \uparrow i_q : p \rightarrow \uparrow \bigsqcup \{i_1, \ldots, i_q\} : p$$

for $q > 2$.

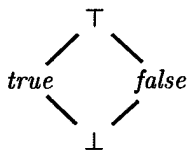
Lemma 11. *Let Γ be a regular Horn formula over a signature Σ, let I^* be a classical interpretation satisfying Γ^*, let $p \in \Sigma$, and let $M \subset N$ be a non-empty set of truth values such that, for all $j \in M$,*

$$I^*(\uparrow j : p) = \mathit{true} \;;$$

then

$$I^*(\uparrow \bigsqcup M : p) = \mathit{true} \;.$$

Example 2. Assume that $N = \{\bot, \top, \mathit{true}, \mathit{false}\}$ and the partial order over N is the lattice shown below:



Given a regular Horn formula Γ (over signature Σ), for each propositional variable p occurring in Γ we add the following classical Horn clauses (over the signature Σ^*) to obtain Γ^*:

$$\begin{array}{ll}
(D^*_1) & \uparrow \top : p \rightarrow \uparrow \mathit{true} : p \\
(D^*_2) & \uparrow \top : p \rightarrow \uparrow \mathit{false} : p \\
(D^*_3) & \uparrow \mathit{true} : p \rightarrow \uparrow \bot : p \\
(D^*_4) & \uparrow \mathit{false} : p \rightarrow \uparrow \bot : p \\
(E^*_1) & \uparrow \mathit{true} : p, \uparrow \mathit{false} : p \rightarrow \uparrow \top : p
\end{array}$$

The following theorem states the correctness of the transformation *:

Theorem 12. *A regular Horn formula Γ is satisfiable over some truth-value lattice $(N, \geq)$ if and only if Γ^* is satisfiable.*

4.2 Results

The size of Γ^* is easily seen to have

$$|\Gamma| + 3s|N|^2$$

as an upper bound where s is the number of different atoms occurring in Γ.

As $s < |\Gamma|$, this places $|\Gamma^*|$ in $\mathcal{O}(|\Gamma||N|^2)$; and, since the time complexity of constructing Γ^* is linear in its size, the reduction * is in $\mathcal{O}(|\Gamma||N|^2)$.

If the ordering $\geq$ on the truth value set is total, the size of Γ^* is bounded by

$$|\Gamma| + 2s|N| \ ,$$

as in that case there are only $|N|$ many clauses D_{ijk} for each k in Γ^*, and no clauses E_{ijk} are needed. Then, the transformation * is in $\mathcal{O}(|\Gamma||N|)$.

Theorem 13. *The transformation * is computable in time linear in the size of the transformed formula and quadratic in the size of the truth value set.*

If the ordering $\geq$ is total, it is linear in both the size of the transformed formula and the size of the truth value set.

Because classical Horn SAT is solvable in linear time [2], we obtain the following corollaries (for distributive lattices similar results were proven in [16]):

Corollary 14. *Regular Horn SAT can be solved in time linear in the size of the formula and quadratic in the size of the truth value lattice.*

Corollary 15. *For all fixed truth-value lattices $(N, \geq)$, regular Horn SAT$_{(N,\geq)}$ can be solved in linear time.*

In the special case of totally ordered truth values, regular Horn SAT is of even smaller complexity (which was already known, see [7]).

Corollary 16. *Regular Horn SAT with a* totally *ordered set of truth values can be solved in time linear in both the size of the formula and the size of the truth value set.*

4.3 Regular Unit Resolution

In this subsection, we define a regular unit resolution calculus and state its completeness for regular Horn clauses. The calculus is based on the inference rules shown in Table 1.

Theorem 17. *A regular Horn formula Γ is unsatisfiable if and only if there exists a derivation of the empty clause from Γ using the calculus formed by the PRUR rule and the RR rule.*

Proof sketch. Theorem 12 states that Γ is satisfiable iff Γ^* is satisfiable. Thus, we know that Γ is unsatisfiable iff there exists a derivation of the empty clause from Γ^* using classical positive unit resolution (PUR), since this rule is refutation complete for classical Horn formulae. One proves, by induction on the number n of deduction steps in that derivation using one of the additional clauses that are in Γ^* but not in Γ, that it is possible to construct from a classical (PUR) derivation of the empty clause from Γ^* a deduction of the empty clause from Γ using the PRUR and RR rules. □

Our regular reduction rule can be seen as an improvement of the rule presented in [8]. Whereas they provide a top-down, Prolog-like proof procedure, we have defined a bottom-up procedure based on unit resolution. An alternate solution with an extended notion of signs that avoids reduction rules altogether can be found in [9]. In [8, 9], however, complexity issues are not discussed.

4.4 Infinite Truth Value Lattices

The results of this section so far have only been proven for *finite* truth value lattices; for example, it is essential for Lemma 11 to hold that the set of truth values is finite.

Nevertheless, the results apply in many cases to *infinite* truth value lattices as well, because it suffices to consider the sub-lattice that is generated by the truth values actually occurring in a formula and the bottom element.

Definition 18. Given a regular Horn formula Γ over a (possibly infinite) truth value lattice $(N, \geq)$, we define $(N_\Gamma, \geq)$ to be the sub-lattice of $(N, \geq)$ generated by the elements in

$$\{i \in N \mid i \text{ occurs in } \Gamma\} \cup \{\bot\} \ .$$

The following theorem states that if the satisfiability of a formula Γ is to be checked, it suffices to only consider the truth value lattice $(N_\Gamma, \geq)$. Thus, if N_Γ is finite and effectively computable for all Γ, then all previous results of this section can be made use of by considering the lattice $(N_\Gamma, \geq)$ instead of $(N, \geq)$.

Theorem 19. *Let Γ be a regular Horn formula over a (possibly infinite) truth value lattice $(N, \geq)$. The formula Γ is satisfiable by an interpretation over the lattice $(N, \geq)$ if and only if it is satisfiable over the lattice $(N_\Gamma, \geq)$.*

Proof. The if-part of the theorem is trivially true, because every interpretation over $(N_\Gamma, \geq)$ is an interpretation over $(N, \geq)$ as well.

To prove the only-if part, assume that the N-interpretation I satisfies Γ. Define the N_Γ-interpretation I_Γ for all atoms $p \in \Sigma$ by $I_\Gamma(p) = \bigsqcup M_p$ where

$$M_p = \{i \in N_\Gamma \mid I(p) \geq i\} \ .$$

It suffices to show that for all truth values i occurring in Γ (and thus in N_Γ): $I \models \uparrow i : p$ if and only if $I_\Gamma \models \uparrow i : p$.

a. Assume that $I \models \uparrow i : p$, i.e., $I(p) \geq i$. In that case, $i \in M_p$ and, by definition of I_Γ, we have $I_\Gamma(p) \geq i$ and, thus, $I_\Gamma \models \uparrow i : p$.

b. Assume that $I_\Gamma \models \uparrow i : p$, i.e., $I_\Gamma(p) \geq i$. Since $I(p) \geq j$ for all $j \in M_p$, we have

$$I(p) \geq \bigsqcup M_p = I_\Gamma(p) \geq i$$

and, thus, $I \models \uparrow i : p$ (note that the supremum operator $\sqcup$ is the same in both lattices). □

Positive Regular Unit Resolution (PRUR)

$$\frac{\rightarrow \uparrow i : p \qquad \uparrow i_1 : p_1, \ldots, \uparrow i_l : p, \ldots, \uparrow i_k : p_k \rightarrow \uparrow j : q}{\uparrow i_1 : p_1, \ldots, \uparrow i_{l-1} : p_{l-1}, \uparrow i_{l+1} : p_{l+1}, \ldots, \uparrow i_k : p_k \rightarrow \uparrow j : q}$$

provided that $i \geq i_l$.

Regular Reduction (RR)

$$\frac{\rightarrow \uparrow i : p \qquad \rightarrow \uparrow j : p}{\rightarrow \uparrow (i \sqcup j) : p}$$

provided that neither $i \geq j$ nor $j \geq i$.

Table 1. Inference rules of the regular unit resolution calculus.

Since the formula Γ is finite, the set of elements generating $(N_\Gamma, \geq)$ is finite as well. Therefore, the sub-lattice $(N_\Gamma, \geq)$ is finite if $(N, \geq)$ is *locally finite*, i.e., if every sub-lattice generated by a finite subset is finite. This is, for instance, the case if the lattice $(N_\Gamma, \geq)$ is distributive.

4.5 Extension to Partial Orders with Maximum

One of the main advantages of our transformational approach to signed logic is that it becomes completely transparent which additional deductive machinery is required as compared to the classical case. This becomes clearer even when we go beyond lattice-ordered regular Horn formulae.

We start with two considerations that somewhat limit the terrain. A core feature of any efficient deduction procedure for Horn formulae is the possibility to represent the conjunction of two unit clauses as a single unit clause as witnessed by the reduction rule in the previous section. This amounts to saying that signs of atoms must be closed under conjunction. When signs are upsets this condition can be expressed as:

$$\begin{array}{l}\text{For all } i, j \in N \text{ there is a } k \in N \text{ such that} \\ \uparrow i \cap \uparrow j = \uparrow k\end{array} \qquad (1)$$

Proposition 20. *Every non-empty, finite poset that satisfies (1) is an upper semi-lattice.*

Therefore, it is inevitable to generalise the language of signs if we want to go beyond lattices. A natural candidate for an enriched language of signs are finite unions of upsets which can also be seen as finitely generated filters. In the following we write $\uparrow \{i_1, \ldots, i_k\}$ instead of $\uparrow i_1 \cup \cdots \cup \uparrow i_k$ and similar for $\downarrow$. We extend our notion of regularity (and hence of Horn formulae) as follows:

Definition 21. If a sign S is of the form $\uparrow \{i_1, \ldots, i_k\}$ or $\downarrow \{i_1, \ldots, i_k\}$ for some $\{i_1, \ldots, i_k\} \subseteq N$ and $k \geq 1$, then it is called a *regular sign*.

A signed clause is called *regular* if it contains regular atoms with signs only of the form $\uparrow \{i_1, \ldots, i_k\}$.[3] A signed CNF formula is called regular if it only contains regular clauses. A regular clause containing at most one regular atom with positive polarity is a *regular Horn clause*. A regular CNF formula consisting solely of regular Horn clauses is a *regular Horn formula*.

The next question is which partial orders can be captured if we want to retain an efficient decision procedure for regular Horn formulae. One such necessary condition is that there must be a maximum $\top$. To see this consider $N = \{false, true\}$ with $false \not\leq true$ and $true \not\leq false$. In this case $\{false\} = \uparrow \{false\}$ and $\{true\} = \uparrow \{true\}$, so each classical CNF clause can be expressed as a regular Horn clause. Given

$$C = p_1, \ldots, p_k \rightarrow q_1, \ldots, q_l \ ,$$

rewrite C, for example, into:

$$\uparrow \{true\} : p_1, \ldots, \uparrow \{true\} : p_k, \uparrow \{false\} : q_1, \ldots, \uparrow \{false\} : q_{l-1} \rightarrow \uparrow \{true\} : q_l$$

Hence, we cannot expect to obtain a polynomial decision procedure for such Horn formulae. The problem is that by conjoining regular signs $\uparrow false$ and $\uparrow true$ we can express falsity at any time on the object level which is as good as to admit contrapositives of clauses.

Finally, we sketch how partial orders with maximum lead to a reasonable notion of generalised Horn formulae. This can be done via a reduction to lattice-ordered Horn formulae handled in the previous sections. For a partial order $(N, \leq)$ with maximum $\top$ consider the lattice $\mathcal{F}^+(N)$ of its non-empty order filters. Its elements can be represented as the non-empty anti-chains of $(N, \leq)$, that is

$$\{S \mid \emptyset \neq S \subseteq N, \text{ and for all } i, j \in S\text{: if } i \neq j \text{ then } i \not\leq j, j \not\leq i\} \ .$$

The order $\sqsubseteq$ on $\mathcal{F}^+(N)$ is defined as $S \sqsubseteq S'$ iff $\Uparrow S \supseteq \Uparrow S'$, where $\Uparrow S$ is the filter generated by S in N. To apply the results of the previous sections it is sufficient to show:

Proposition 22. *A regular literal $\uparrow S : p$ is satisfiable w.r.t. a poset with maximum $(N, \leq)$ iff it is satisfiable w.r.t. the lattice $\mathcal{F}^+(N)$.*[4]

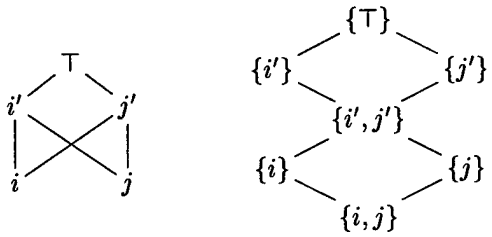
[3] The remark made at the end of Section 2.2 applies here as well.

[4] In the latter case, of course, S is interpreted as a single lattice element in $\mathcal{F}^+(N)$.

There is a price to pay for the increased generality: The lattice $\mathcal{F}^+(N)$ can be considerably larger than the poset $(N, \leq)$, in the worst case exponentially larger. This proves:

Theorem 23. *Regular Horn SAT formulae based on posets with maximum can be solved in time linear in the size of the formula and exponential in the size of the truth value set.*

Example 3. Consider the poset N depicted below on the left. The lattice $\mathcal{F}^+(N)$ is shown on the right. It can be seen as a lattice-completion of N.



From a deductive point of view it is important to compute the supremum $\sqcup$ and $\sqsubseteq$ in $\mathcal{F}^+(N)$, because these are required in the reduction and unit resolution rule, respectively. This is done as follows:

Let $\Uparrow S = \Uparrow\{i_1, \ldots, i_k\}$ and $\Uparrow S' = \Uparrow\{j_1, \ldots, j_l\}$ be given. We denote with $max(i, j)$ the set of minimal elements above i and j in N w.r.t. $\leq$. Now $S \sqcup S' = \Uparrow S \cap \Uparrow S' = \{k \mid k \in \max(i, j),\ i \in S,\ j \in S'\}$. From the resulting set any elements not minimal in it can be deleted to arrive at an anti-chain representation. Finally, $S \sqsubseteq S'$ iff $\Uparrow S \supseteq \Uparrow S'$ iff for all $j_r \in S'$ there is a $i_s \in S$ such that $i_s \leq j_r$.

5 Future Work

An investigation of the lattice theoretic aspects of lattice-ordered regular Horn formulae could lead to useful new results. In particular, in the infinite case, for which only first ideas have been presented in Section 4.4, representation theory and dualities should be further studied. Priestley duality has already been successfully exploited in the case of distributive lattices [15].

As another line of work, experiments should be carried out to compare our tailored decision procedures for regular Horn formulae with procedures for classical Horn formulae after applying the transformation * defined in Section 4.

Acknowledgement

We thank Viorica Sofronie-Stokkermans for fruitful discussions and useful comments on an earlier version of this paper. The comments of an anonymous referee helped to clarify the presentation.

References

[1] S. Cook. The complexity of theorem-proving procedures. In *Proceedings, 3rd Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[2] W. Dowling and J. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulæ. *J. of Logic Programming*, 3:267–284, 1984.

[3] G. Escalada-Imaz and F. Manyà. The satisfiability problem for multiple-valued Horn formulæ. In *Proc., International Symposium on Multiple-Valued Logics (ISMVL), Boston, USA*, pages 250–256. IEEE Press, Los Alamitos, 1994.

[4] G. Escalada-Imaz and F. Manyà. Efficient interpretation of propositional multi-valued logic programs. In *Proceedings, International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU), Paris, France*, LNCS 945, pages 428–439. Springer, 1995.

[5] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM J. of Computing*, 5:691–703, 1976.

[6] R. Hähnle. *Automated Deduction in Multiple-Valued Logics*. Oxford University Press, 1994.

[7] R. Hähnle. Exploiting data dependencies in many-valued logics. *J. of Applied Non-Classical Logics*, 6(1):49–69, 1996.

[8] M. Kifer and V. S. Subrahmanian. Theory of generalized annotated logic programming and its applications. *J. of Logic Programming*, 12:335–367, 1992.

[9] S. M. Leach and J. J. Lu. Query processing in annotationed logic programming: Theory and implementation. *J. of Intelligent Information Systems*, 6(1):33–58, 1996.

[10] J. J. Lu. Logic programming with signs and annotations. *J. of Logic and Computation*, 6(6):755–778, 1996.

[11] J. J. Lu, J. Calmet, and J. Schü. Computing multiple-valued logic programs. *Mathware & Soft Computing*, IV(2):129–153, 1997.

[12] J. J. Lu, L. J. Henschen, V. S. Subrahmanian, and N. C. A. da Costa. Reasoning in paraconsistent logics. In R. Boyer, editor, *Automated Reasoning: Essays in Honor of Woody Bledsoe*. Kluwer, 1991.

[13] J. J. Lu, N. V. Murray, and E. Rosenthal. Signed formulas and annotated logics. In *Proceedings, International Symposium on Multiple-Valued Logics (ISMVL)*, pages 48–53. IEEE Press, Los Alamitos, 1993.

[14] F. Manyà. *Proof Procedures for Multiple-Valued Propositional Logics*. PhD thesis, Facultat de Ciències, Universitat Autònoma de Barcelona, 1996.

[15] V. Sofronie-Stokkermans. *Fibered Structures and Applications to Automated Theorem Proving in Certain Classes of Finitely-Valued Logics and to Modeling Interacting Systems*. PhD thesis, Universität Linz, Forschungsinstitut für symbolisches Rechnen, 1997.
`http://www.mpi-sb.mpg.de/~sofronie`.

[16] V. Sofronie-Stokkermans. On translation of finitely-valued logics to classical first-order logic. In *Proceedings, 13th European Conference on Artificial Intelligence (ECAI)*, pages 410–411, 1998.

Semirigidity Problems in k-Valued Logic

Masahiro Miyakawa
Tsukuba College of Technology
Tsukuba-shi, Japan 305-0821
mamiyaka@cs.k.tsukuba-tech.ac.jp

Abstract

The study of semirigid sets arose from the classification of bases. In this complex problem - fully solved only for $|A| = 2, 3$ - one of the tasks is to find all minimal nontrivial intersections of systems of maximal clones. Most of the clones are determined by reflexive relations (binary or of higher arities) and so we need to determine subsets R of these relations such that every function preserving all relations in R is either constant or is a projection.

In this paper we give a short overview of this problem for 1) isotone relations, 2) central relations and 3)quasi-linear relations. Finally we add some new results for 4) autodual clones.

1. Introduction

In the sequel the *universe* A is a fixed nonempty set. For a positive integer n denote by $O_A^{(n)}$ the set of all n-ary *operations* or functions on A (i.e. maps f: $A^n \to A$) and put $O_A := \bigcup_{n=1}^{\infty} O_A^{(n)}$. For example, for $1 \leq i \leq n$ the n-ary operation e_i^n on A, called the i-th n-ary *projection*, is defined by setting $e_i^n(a_1, \ldots, a_n) := a_i$ for all $a_1, \ldots, a_n \in A$. Roughly, a *clone* A is a composition closed subset of O_A containing the set J_A of all projections, or, equivalently, the set of term operations of a universal algebra $\langle A; F\rangle$ (where $F \subseteq O_A$). For a more precise definition cf. [8] or [20]. For example, the sets J_A and O_A are clones. Also the set K_A consisting of all projections and all constant operations (i.e. $f \in O_A^{(n)}$ with $|im\, f| = 1$, i.e. taking a single value) is a clone. The clones on A, ordered by inclusion, form a lattice L_A with the least element J_A and the greatest element O_A in which the meet of any family of clones is their set theoretic intersection. Let A be finite and $|A| > 2$. Then $|L_A| = 2^{\aleph_0}$ [3] and the lattice is largely unknown. The dual atoms (= coatomes, i.e. elements covered by O_A), called *maximal* or *precomplete* clones, are explicitly known. Moreover, each proper clone is a subclone of a maximal clone. For their description we need the following concept.

Let h be a positive integer . A subset ρ of A^h (i.e. a set of h-tuples over A) is an h-ary *relation* on A. An n-ary $f \in O_A$ *preserves* ρ if for every $h \times n$ matrix $X = [x_{ij}]$ over A whose columns are all h-tuples from ρ we have

$$(f(x_{11}, \ldots, x_{1n}), f(x_{21}, \ldots, x_{2n}), \ldots, f(x_{h1}, \ldots, x_{hn})) \in \rho$$

(i.e. ρ is a subuniverse of the h-th power $\langle A; f\rangle^h$ of $\langle A; f\rangle$). For example, if ρ is a (partial) order $\leq$ (i.e. a binary reflexive, antisymmetric and transitive relation) then f preserves $\leq$ exactly if f is isotone, i.e. if $f(x_{11}, \ldots, x_{1n}) \leq f(x_{21}, \ldots, x_{2n})$ whenever $x_{11} \leq x_{21}, \ldots, x_{1n} \leq x_{2n}$. The set of all $f \in O_A$ preserving a given relation ρ is denoted by Pol ρ. There are 6 well-described finite families $\mathcal{R}_1, \ldots \mathcal{R}_6$ of relations on A such that $\{\text{Pol } \rho : \rho \in \mathcal{R}_1 \cup \ldots \cup \mathcal{R}_6\}$ are precisely the maximal clones (cf. [20]). The numbers of maximal clones are known (cf. [19]). The most rich families are those that preserve order relations and central relations. A remarkable fact is that the intersection of all maximal clones is the least element J_A of L_A. Call an h-ary relation on ρ on A *reflexive* if $(a, \ldots, a) \in \rho$ for all $a \in A$. The nonreflexive relations in $\mathcal{R} := \mathcal{R}_1 \cup \ldots \cup \mathcal{R}_6$ are all the unary relations ρ (i.e. subsets of A) such that $\phi \neq \rho \neq A$ and relations of the form $\{(a, \pi(a)) : a \in A\}$ where π is a special permutation on A (investigated in Sections 4). It is immediate that $K_A \subseteq Pol\ \rho$ for every reflexive relation ρ on A. Call a set $\{\rho_i : i \in I\}$ of reflexive relations on A *semirigid* if $K_A = \bigcap_{i \in I} Pol\ \rho_i$. The question is to determine semirigid subsets of $\mathcal{R}$. A unary operation f on A (i.e. a selfmap of A) is a *joint endomorphism* of a set R of relations on A if $f \in O_A^{(1)} \cap \bigcap_{\rho \in R} Pol\ \rho$. The set of joint endomorphism of R is denoted by End R. It is known (cf. Lemma ?? below) that R is semirigid if and only if End $R = K_A^1 = O_A^{(1)} \cap K$. Apparently, the first known example of a semirigid set of relations is the set θ_A of all (binary) equivalence relations on A (cf. [15, Ch.1, Probl. 2,p.38]). Two clones are of a same *type* if the relations they preserve are isomorphic. It is natural to consider semirigidity first within a type. As there are variety of types, we shortly

0-7695-0161-3/99 $10.00 © 1999 IEEE

overview semirigid sets of relations on A. As a special case (also for simplicity) a relation ρ is *semirigid* (C-rigid [5] or endorigid [17]) if End $\rho = K_A^{(1)}$.

2. Reduction to unary functions

Let C be a clone on the universe A. Call $C^{(1)} := C \cap O^{(1)}$ (i.e. the set of unary operations from C) the *foundation* of C (hereafter the index $_A$, denoting the universe, is sometimes omitted; e.g. $O^{(1)}$ denotes $O_A^{(1)}$).

The following lemma is from [5], Prop. 2.2 and also follows from [14].

Proposition 1. *Let $|A| > 2$. Then $K^{(1)}$ is the foundation of a clone C on A if and only if $C = K$.*

The statement is not true for $|A| = 2$ (take e.g. $A = \{0, 1\}$ and the clone generated by the constants and the conjunction $\wedge$). This lemma reduces the semirigidity problem of many-valued logic to the foundations of their clones.

This lemma was strengthened by replacing K by K_{h-1}, where $K_{h-1} := \{f \in O | \, |im\, f| \leq h-1\} \cap J$ for $1 < h \leq k := |A|$ (the case $h = 2$ reduces to Proposition 1) [10]. However, we have:

Proposition 2. *The following statement is not true: for a clone C on A ($|A| \geq 2$ finite)*

$$\text{if } C^{(1)} = J^{(1)} (= \{e\} \text{ :identity map) then } C = J.$$

Proof. We can take e.g. the clone generated by the conjunction $\wedge$, where $x \wedge y := \min(x, y)$ with respect to the natural order $0 < 1 \ldots < k-1$ (where $|A| = k$). □

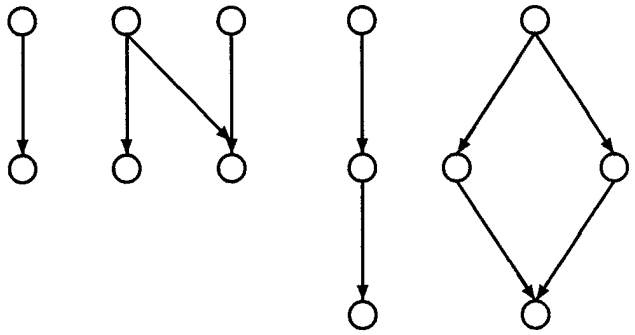
In Section 4 we formulate a special (weaker) case of this statement as an open problem.

3. A brief overview

Order relations

For an order $\leq$ on A the clone Pol $(\leq)$ is maximal if and only if the order is bounded (= it has a least element o and a greatest element e, i.e. $o \leq a \leq e$ holds for all $a \in A$) [9]. The maximal clones of the form Pol $(\leq)$ are exceptional in the sense that they are not be necessarily finitely generated [21]. It seems that the characterization of semirigid set of orders is not easy. Three types of orders has been considered so far 1) the orders of height 1, 2) the chains and 3) so called the diamonds (Fig. 1).

In [2] a fairly general approach is taken successfully to this problem in the framework of order relations (not necessarily bounded). They addressed the following problem: What is the least m such that there are orders $\leq_1, \ldots, \leq_m$



1) height 1 (fence) 2) chain 3) diamond

Figure 1. Orders

on A having only trivial joint endomorphism (i.e. End $\{\leq_1, \ldots, \leq_m\} = K_A^{(1)}$). In the paper semirigid binary relations ρ (i.e. an oriented graph $G = (A, \rho)$) are constructed systematically for $|A| > 7$ or $|A| = 6$. Based on a semirigidity criterion (Theorem 2.4 [5]) considerable effort is devoted to the semirigidity proof of the graphs. Then the ρ is represented as a (set theoretical) union of two orders $\leq$ and $\leq'$ both of height 1 (an order $(A, \leq)$ is of *height* 1 if each $a \in A$ is either minimal or maximal). We have

Lemma 3. *Let R be a set of binary relations on a set A. Then the least set R^* of binary relations on A containing R and closed with respect to (i) arbitrary (set-theoretical) unions and intersections and (ii) inverses satisfies*

$$\text{End } R \subseteq \text{End } R^*.$$

From this we can conclude

$$\text{End } \{\leq, \leq'\} \subseteq \text{End } \rho = K_A^{(1)}.$$

For example, for a graph G on 6 in Fig.2,

$$\begin{aligned} \leq &= \{(0,1),(2,3),(4,5)\} \cup \iota_A \text{ and} \\ \leq' &= \{(1,2),(1,4),(3,0),(5,0)\} \cup \iota_A \end{aligned}$$

where $\iota_A = \{(a, a) | a \in A\}$.

The general construction fails for $|A| < 6$. The exception for $|A| = 7$ can be fixed separately. For $|A| = 3, 4$ there are no semirigid set of binary relations [1] and they must be considered separately. This may be fixed by case analysis. For the semirigidity of chains and diamonds the situations were settled completely. For $|A| = 3$ we need at least 3 chains while for $|A| > 3$ there are semirigid pairs of chains. For diamond orders such minimum number is 4 for $|A| = 4, 5$ (for $|A| = 3$ the diamond reduces to a chain).

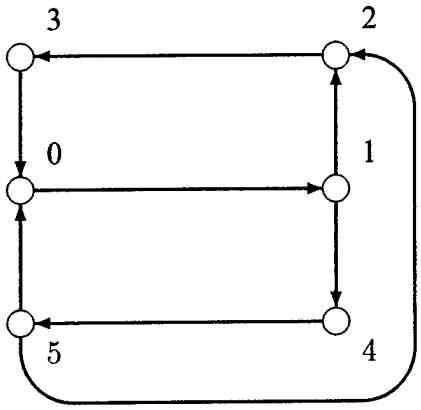


Figure 2. A semirigid graph G ($|A| = 6$)

In [11] the problem is considered for chains (= linear orders). A simple criterion for semirigidity is given. For $A = \boldsymbol{k} = \{0, 1, \ldots, k-1\}$ where $k > 3$ there always exists a chain which together with the natural order $0 < 1, \ldots < k-1$ forms a semirigid system. For example, the chain $1 \prec 3 \prec 0 \prec 2$ is the only chain such that for $k = 4$:

$$0 < 1 < 2 < 3 \text{ and } 1 \prec 3 \prec 0 \prec 2$$

form a semirigid system.

It is shown that the proportion of such chains in the total of $k!$ chains tends to $1/e = 0.13\ldots$ as k becomes large. In [7] the problem is considered for diamond orders. An order relation $\leq_{ab}$ is a *diamond* provided $x \leq_{ab} y$ holds exactly if $x = a$ or $y = b$. The semirigidity criterion is given. For example, $\{\leq_{01}, \leq_{03}, \leq_{12}, \leq_{13}\}$ and $\{\leq_{01}, \leq_{23}, \leq_{24}, \leq_{34}\}$ are the semirigid systems for $k = 4$ and $k = 5$, respectively.

The number of semirigid sets of diamonds is also given explicitly for all $k \geq 4$.

Central relations

For a set of h-ary central relations ($1 < h \leq k := |A| > 2$) the semirigidity problem is naturally extended[10] to :

A set of h-ary central relations on a set A is called *semirigid* if the clones of k-valued logic functions determined by the relations share only the clone K_{h-1} consisting of all projections and all functions assuming at most $h-1$ values ($1 < h \leq k := |A| > 2$). For $h = 2$ this definition reduces to the semirigidity described before.

It is shown that the minimum size of a semirigid set of h-ary central relations is $h + 1$ [10]. The paper [12] has some additional results, but no systematic study has been carried out.

Quasi-linear relations

In [13] the semirigidity problem is considered for the so called *quasi-linear* (or *affine*) clones for k prime. For $k = 5$ we need at least 3 such clones to be semirigid, while for $k > 5$ prime there always exist a semirigid pair of clones. The case k power of a prime has not yet been investigated (the notion of quasi-linear functions becomes complicated).

4. Autodual clones

For areflexive relations the semirigidity problem reduces to the *rigidity* problem because they do not admit any constant functions (thus, a set $\{\rho_i : i \in I\}$ of areflexive relations on A is *rigid* if $J_A = \bigcap_{i \in I} Pol\ \rho_i$). The situation seems to be much simpler. We investigate it here. Set $A := \boldsymbol{k} := \{0, \ldots, k-1\}$ and let S_k denote the symmetric group over $\boldsymbol{k}$.

For a permutation $s \in S_k$ set

$$s^{\square} = \{(x, s(x)) | x \in \boldsymbol{k}\}.$$

The operations of the clone $\mathrm{Pol}\, s^{\square}$ is called *autodual* with respect to s. It is known that $\mathrm{Pol}\, s^{\square}$ is maximal if and only if the permutation s is the product of ℓ cycles of the length p, where $k = p \cdot \ell$ and p is a prime number. In this section we fix such a permutation s.

For $s \in S_k$ call $\{s, s^2, \ldots, s^k\}$, where $s^k = e$ *orbit* of s. Two permutations are *orbit-free* if their orbits share only the identity function e.

The following lemma is known (cf. [6]):

Lemma 4. *For $s, s' \in S_k$ we have $\mathrm{Pol}\, s^{\square} = \mathrm{Pol}\, s'^{\square}$ if and only if*

$$s^i = s' \text{ for some } i \in \{1, \ldots, k-1\};$$

in other words, two permutations yield distinct autodual clones if and only if they are orbit-free.

For $k = \ell \cdot p$ (p prime, $\ell \geq 1$) and $\boldsymbol{k} = \{a_{1,0}, \ldots, a_{1,p-1}, a_{2,0}, \ldots, a_{2,p-1}, \ldots, a_{\ell,0}, \ldots, a_{\ell,p-1}\}$ let

$$s(a_{i,r}) = a_{i,r+1 \pmod p}\ (i = 1, \ldots, \ell; r = 0, \ldots, p-1). \tag{1}$$

This gives

$$s^j(a_{i,r}) = a_{i,r+j \pmod p}) \text{ for } j = 0, 1, \ldots.$$

As elements of $\{a_{i,0}, \ldots, a_{i,p-1}\}$ (i-th component) are shifted cyclically by s, we may assume $a_{i,0} = \min\{a_{i,0}, \ldots, a_{i,p-1}\}$.

From the definition of $s^{\square}$ a unary function f is autodual with respect to s if and only if

$$s(f(x)) = f(s(x)). \tag{2}$$

This yields

$$f(a_{i,r}) = s^r(f(a_{i,0}))\ (r = 0, \ldots, p-1; i = 1, \ldots, \ell). \tag{3}$$

To represent a unary function $f \in \operatorname{Pol} s^{\square}$, let us call the set $\{a_{i,0}, \ldots, a_{i,p-1}\}$ i-th *cycle* of f and call a restriction of f to the i-th cycle (a map from the i-th cycle into $\boldsymbol{k}$) i-th-*component* of f. Denote the i-th component of f by

$$\begin{bmatrix} a_{i,0}, & \ldots, & a_{i,p-1} \\ f(a_{i,0}), & \ldots, & f(a_{i,p-1}) \end{bmatrix}.$$

We have the following component-wise representation for a unary autodual function f.

Theorem 5. *A unary function f is autodual with respect to s if and only if its i-th component is of the form*

$$\begin{bmatrix} a_{i,0} & , \ldots, & a_{i,p-1} \\ a_{j_i,r_i} & , \ldots, & a_{j_i,r_i+p-1} \pmod p \end{bmatrix}.$$

for each $i \in \{1, \ldots, \ell\}$, where $j_1, \ldots, j_\ell \in \{1, \ldots, \ell\}$ and $r_1, \ldots, r_\ell \in \{0, \ldots, p-1\}$. In other words, f should map every cycle onto a cycle preserving s.

Proof. ($\Leftarrow$)We show that f defined by $f(a_{i,m}) := a_{j_i,r_i+m \pmod p}$ for each component $i \in \{1, \ldots, \ell\}$ and $m \in \{0, \ldots, p-1\}$ satisfies (2). Indeed, by (1) $s(a_{i,m}) = a_{i,m+1 \pmod p}$, so

$$\begin{aligned} f(s(a_{i,m})) &= a_{j_i,r_i+m+1 \pmod p} \\ &= s(a_{j_i,r_i+m \pmod p}) = sf(a_{i,m}). \end{aligned}$$

($\Rightarrow$) Assume that $f$satisfies (2). Let $1 \le i \le \ell$ and $f(a_{i,0}) = a_{j_i,r_i}$ for some $j_i \in \{1, \ldots, \ell\}$ and $r_i \in \{0, \ldots, p-1\}$. Then by (3) clearly $f(a_{i,r}) = a_{j_i,r_i+r}$ and the i-th component of f has the required form. □

We have the following corollaries:

Corollary 6. *The number of unary functions that autodual with respect to s is*

$$|(\operatorname{Pol} s^{\square})^{(1)}| = k^{\ell}. \tag{4}$$

Proof. We have ℓ choices for j_i and p choices for r_i, and this can be done for each $i = 1, \ldots, \ell$. □

Corollary 7. *The number of permutations on $\boldsymbol{k}$ autodual with respect to s is*

$$|(\operatorname{Pol} s^{\square})^{(1,\text{onto})}| = \ell!\, p^{\ell}. \tag{5}$$

Proof. The only difference is that the choice for j_i should be such that $j_1, \ldots, j_\ell$ should be a permutation of $\{1, \ldots, \ell\}$. So we have a total of $\ell!$ choices for the set $\{j_1, \ldots, j_\ell\}$. For r_i the situation remains the same as in the preceding corollary. □

From Theorem 5 we have the following results on the rigidity problem for autodual clones.

Corollary 8. *For k prime the unary functions of every pair of autodual maximal clones shares exactly the identity function e.*

Proof. By Theorem 5 the set of unary operations of Pol $s^{\square}$ is exactly its orbit $\{s, s^2, \ldots, s^p\}$. By Lemma 4 distinct maximal clones are induced exactly by orbit-free permutations. Thus they share exactly the identity function e. □

The following result is also immediate from Theorem 5.

Corollary 9. *Let two permutations s and s' share a component. Then $\{s^{\square}, s'^{\square}\}$ is non-rigid.*

However, the next example shows that the converse is not true; i.e. it is not sufficient for rigidity that s and s' share no component.

Example 1. Let s_1 and s_2 be

$$s_1 = \begin{pmatrix} 01\ 23 \\ 10\ 32 \end{pmatrix} \text{ and } s_2 = \begin{pmatrix} 02\ 13 \\ 20\ 31 \end{pmatrix}.$$

It can be verified that $s_3 = \left(\begin{smallmatrix} 03\ 12 \\ 30\ 12 \end{smallmatrix}\right)$ belongs to $\operatorname{Pol} s_1^{\square} \cap \operatorname{Pol} s_2'^{\square}$.

The following example shows that even two permutations with different prime cycle-lengths may share a non-trivial unary function.

Example 2. Both clones

$$\operatorname{Pol} \begin{pmatrix} 01\ 23\ 45 \\ 10\ 32\ 54 \end{pmatrix} \text{ and } \operatorname{Pol} \begin{pmatrix} 012\ 345 \\ 120\ 453 \end{pmatrix}$$

admit the non-trivial unary function $\begin{bmatrix} 012345 \\ 453201 \end{bmatrix}$.

We have considered unary functions of autodual clones. We formulate some open problems in order to solve the rigidity problem.

Let P denote the set of prime divisors of k. Denote by Σ the set of all permutations of $\boldsymbol{k} = \{0, \ldots, k-1\}$ with k/p cycles of length p where p runs through P.

Question 1. Suppose that $s_1, \ldots, s_m \in \Sigma$ are such that $(\cap_{i=1}^m \operatorname{Pol} s_i^{\square})^{(1)} = \{e\}$. Is it true that $\cap_{i=1}^m \operatorname{Pol} s_i^{\square} = J$ (= the clones of all projections)?

If the answer is yes, then Corollary 8 means that every autodual maximal clones is rigid for k prime. One should start looking at $k = 5$. Indeed for $k = 3$ there is only one clone Pol $s^{\square}$ with $s \in \Sigma$. For $k = 4$ the set Σ is $\{s_1, s_2, s_3\}$ given in Example 1 whereby $s_i \in \operatorname{Pol} s_1{\square} \cap \operatorname{Pol} s_2{\square} \cap \operatorname{Pol} s_3{\square}$ $(i = 1, 2, 3)$ and therefore $\{s^{\square} | s \in \Sigma\}$ is not rigid.

Question 2. Characterize rigid $\{s_1^{\square}, \ldots, s_m^{\square}\}$ with $s_1, \ldots, s_m \in \Sigma$.

Again, one should start looking at cyclic orbit-free permutations of 5.

5. Conclusions

We briefly overviewed the semirigidity problems. We conjectured that autodual clones are rigid to each other for k prime. Unfortunately, we could not solve the problem completely. Instead, we listed some open questions.

For the constant-preserving maximal clones (unary relations corresponding to a non-empty proper subset $\emptyset \neq \rho \neq A$), the trivial k-system of relations $\{\{0\}, \ldots, \{k-1\}\}$ is rigid and k seems to be the minimum number of cardinality for such a system.

Several types of relations can be investigated further. Among them we include h-regular relations (the only remaining family among $\mathcal{R}$ that is not investigated yet). Semirigidity among different types of relations could be investigated systematically.

Acknowledgement. The author is grateful to Prof. I.G. Rosenberg for his valuable suggestions. The author borrowed many descriptions from [2].

References

[1] V. Chvátal. On finite and countable rigid graphs and tournaments. *Comment. Math. Univ. Caroline* **6**:429-438, 1965.

[2] J.Demetrovics, M.Miyakawa, I.G.Rosenberg, D.Simovici, I.Stojmenović. Intersections of isotone clones on a finite set, *Proc. 20th International Symp. on Multiple-Valued Logic,* Charlotte, 248-253, 1990.

[3] Iu.I.Ianov, A.A.Mučnik. Existence of k-valued closed classes without a finite basis (Russian), *Dokl. Akad. Nauk. SSSR* **127**(1):44-46, 1959.

[4] S.V.Iablonskiĭ. Functional constructions in a k-valued logic (Russian), *Trudy Math. Inst. Steklov* **515**-142, 1958.

[5] F.Länger, R.Pöschel. Relational systems with trivial endomorphisms and polymorphisms. *J. Pure Appl. Algebra* **32**(2):129-142, 1984.

[6] D.Lau. *Funktionenalgebren über endlichen Mengen Band 1*, Monograph, Universität Rostock, pp. 592, 1998.

[7] V.Laskia, M.Miyakawa, A.Nozaki, G.Pogosyan, I.G.Rosenberg. Semirigid sets of diamond orders, *Discrete Mathematics* 156:277-283, 1996.

[8] A.I.Maltsev. *Post's Iterative Algebras* (Russian), Monograph, Novosibirsk State University, 1976.

[9] V.V.Martyniuk. Investigation of certain classes of functions in many-valued logics (Russian). *Problemy Kibernet.* **3**:49-60, 1960.

[10] M.Miyakawa, A.Nozaki, G.Pogosyan, I.G.Rosenberg. Semirigid sets of central relations over a finite domain, *Proc. 22th International Symposium on Multiple-Valued Logic*, 300-307, May, 1992.

[11] A.Nozaki, M.Miyakawa, G.Pogosyan, I.G.Rosenberg. The number of orthogonal permutations, *Europ. J. Combinatorics* **16**:71-85, 1995.

[12] A.Nozaki, G.Pogosyan. Semi-rigid sets of central relations, International Christian University, Preprrint 4, 1992.

[13] A.Nozaki, G.Pogosyan, M.Miyakawa, I.G.Rosenberg. Semirigid sets of quasi-linear clones, *Proc. 23rd International Symp. on Multiple-Valued Logic*, Sacramento, 1993, 105-110.

[14] P.P.Pálfy. Unary polynomials in algebra I. *Algebra Universalis* **18**:162-273, 1984.

[15] R.S.Pierce. *Introduction to the theory of abstract algebras*. Holt, Reinhart and Winston, 145 pp, 1968.

[16] R.Pöschel, L.A.Kalužnin. *Funktionen und Relationen Algebren*, Ein Kapitel der Diskreten Mathematik (German), Math. Monographien B. 15, VEB Deutscher Verlag d. Wissen., Berlin, pp.259, 1979. Also Math. R. B. 67, Birkhäuser Verlag, Basel & Stuttgart, 1979.

[17] M.Pouzet, I.G.Rosenberg. Ramsey properties for classes of relational systems. *Europ. J. Combinatorics* **6**:361-368, 1985.

[18] I.Rival, N.Zaguia. Perpendicular orders, Technical Report Tr-91-10, University of Ottawa, March 1991.

[19] I.G.Rosenberg. The number of maximal closed classes in the set of functions over a finite domain, *J. Combinat. Theory, Ser. A* 14:1-7, 1973.

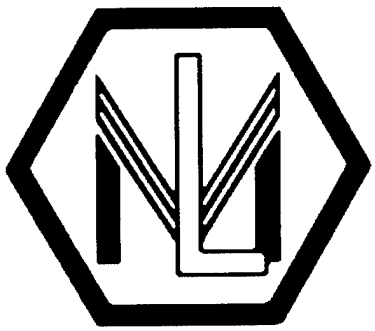
[20] I.G.Rosenberg. Completeness properties of multiple-valued logic algebra, in *Computer Science and Multiple-valued logic, Theory and Applications* (D.C. Rine ed.), North-Holland,s 2-nd revised ed., 144-186, 1984.

[21] G.Tardós. A maximal clone of monotone operations which is not finitely generated, *Order* **3**:211-218, 1986.

SESSION IXA

TESTING

CHAIR: Tsutomu Sasao



Fault Characterization and Testability Considerations in Multi-Valued Logic Circuits

Mostafa Abd-El-Barr, Maher Al-Sherif, and Mohamed Osman
Department of Computer Enginering
King Fahd University of Petroleum and Minerals
Dhahran 31261
Saudi Arabia
mostafa@ccse.kfupm.edu.sa

Abstract

With the growing interest and the emergence of various implementations of Multiple-Valued logic (MVL) circuits, testability issues of these circuits are becoming crucial. Fault characterization is an early step in the test generation process. It is aimed at finding fault models that best describe the possible faults expected to occur in a given class of circuits or technology. Layout and device level studies on CMOS and BiCMOS circuits revealed that the stuck-at model is not adequate to represent the actual physical defects. In this paper, our aim is to characterize faults in a CMOS functionally complete set of MVL operators. The set has been implemented using existing standard binary CMOS technology. This enables us to characterize faults in these operators using the same techniques used for standard binary CMOS. Fault categories in MVL circuits and recommendations for testability will be given.

1 Introduction

It has been recently possible to implement MVL circuits using the available binary technologies, such as the standard CMOS technology. This resulted in various MVL implementations either as stand alone circuits, e. g. multiplier chips, or as modules integrated with binary circuits [5, 6].

With the increasing number of MVL implementations, testing such circuits is becoming crucial. It is more involved to test an MVL circuit as compared to its binary counterpart. This is due to the increased number of logic values and the need to distinguish and eliminate an increased number of logic choices. However, during the test generation process, there exists some flexibility since decisions to assign a certain value to a line are not restricted to 0 or 1 only.

Previous work on MVL testing has been primarily based on the stuck-at fault model. The adequacy of this model was questioned by various studies. For this reason, and due to the increased complexity of test generation of MVL circuits, it is very important to recognize the actual type of faults expected to occur in MVL implementations. This characterization becomes an important input to any test generation algorithm.

In this paper, we study the issue of fault characterization and testability considerations in CMOS MVL circuits. The paper is organized as follows. Section 2 introduces background material and the considered MVL set. Section 3 describes the fault characterization methodology used. In Section 4, characterization results in the MVL sets are presented. Testing for the characterized faults is derived in Section 5. Recommendations for testability of MVL circuits are presented in Section 6. The paper ends with concluding remarks and future work.

2 Background Material

In this paper, we use a functionally complete set of MVL operators [5, 6]. This set has been implemented using CMOS and consists of five operators: *literal*, *complement of literal*, *cycle*, *tsum* and *min*. It has been shown that the introduced set is more efficient in realizing MVL functions in terms of the number of operators used and the overall required circuits [6]. The functionality of the set is verified by HSPICE simulation [5]. We will use the reported set realizations to study the possible types of faults that can occur at the end of a production line. The methods used to model possible faults in binary CMOS studies will be applicable to the above MVL realizations since they are built using the available binary CMOS technology.

Let $R = \{0, 1, 2, ..., r-1\}$ be the set of logic values for an r-valued logic where r is the radix. Assume also that $a, b, \in R$, $a \leq b$ and $k \in \{1, 2, ..., r-1\}$. The MVL

0-7695-0161-3/99 $10.00 © 1999 IEEE

operations used in this paper are: *min, tsum, literal,* and *cycle.* The definitions of those operations can be found in [5].

There have been tremendous efforts to understand and characterize physical defects in ICs and their effect on the circuit behavior. Many studies were conducted on faulty chips taken from actual production lines. In general, to find the effects of defects on the circuit behavior **(fault characterization)**, a defect is inserted in the circuit. Then, simulation is performed to find the circuit behavior under that defect. This process can be done at two different levels, namely at the layout level [1, 4] or at the device level [3, 8]. In the first, a defect is inserted in the layout of a fault free circuit. The new circuit, with the inserted fault, is then extracted from the layout. Next, the resultant circuit is simulated to study its behavior under the inserted fault. In the second approach, shorts and opens are inserted directly to the device-level-representation of the fault free circuit. The behavior of the resultant c! ! ircuit then is studied.

The aim of the research work in this paper is to characterize faults in MVL circuits on the device level. Defects will be inserted into each circuit and simulations, of the resultant circuits, will be conducted. The circuits behavior in the presence of the defects will be classified into fault categories to give statistics on the types of faults expected to be found in these circuits. For this purpose, components of the functionally complete set of MVL operators, introduced earlier, will be used. The methodology used for fault characterization is explained in the next section.

3 Fault Characterization Methodology

In characterizing faults in the considered set, the following steps are used.

1. Defects are inserted on transistor terminals (3 shorts and 3 opens, see Figure 1) and considered one at a time. This will be performed for each operator circuit.
2. SPICE will be used to determine the output of the circuits in the presence of each inserted defect.
3. Simulation results will be analyzed to provide statistics on faults and their occurrences.
4. Recommendations for MVL testability will be made based on the characterization results.

In the simulations, the resistance of the inserted short faults will be varied. Typical resistance values considered for CMOS technology vary from several Ohms up to 4k Ohms. Shorts may occur with higher resistances but with low percentage and very high resistance value that will not affect the circuit behavior [2, 7, 9]. An open is modeled as a 10 Gig. Ohms resistors in parallel with a 0.001pF capacitor between the transistor terminal and the node it is normally connected to. Figure 1 shows the open and short faults along with their models.

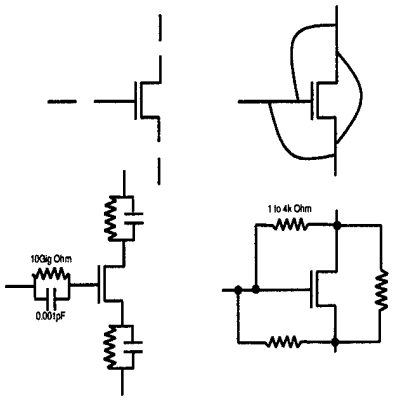


Figure 1. Open and short faults and their models.

The transistors in each circuit will be numbered arbitrarily starting with Current Mode CMOS (CMCL) transistors first then Voltage Mode CMOS (VMCL) transistors. An open at the gate of transistor number 1, for example, will be referred to as O1g. O1d and O1s will refer to opens at transistor 1 drain and source, respectively. S1gs refers to the inserted short between the gate of transistor 1 and its source.

The following terms and abbreviations will be used:

- **SAk** means the output stuck at k under the simulated fault.
- **aSA0** faults in the literals caused the value of a in $k^a[x]^b$ to be as if it was 0. For example, if the original circuit is $k^1[x]^1$, the resultant circuit will be $k^0[x]^1$.
- **bSA3** faults convert the circuit to $k^a[x]^3$. This has no effect if the b in the original circuit was 3.
- **Para_k** fault refers to a parametric change of the value of k. This value is supposed to be the same for the whole input range in [a, b]. Under this fault, k takes several current levels, depending on the input value, which may be read as different logic values. For example, if the circuit $3^1[x]^3$ undergoes this fault, the values of the output current may be different from 60uA (the correct value) for different inputs. For example it could be 15, 40, 60 uA for x = 1, 2 and 3 respectively. This implies that even if the circuit shows the correct output for some input values in the [a, b] range, it could be wrong for others. This implies that the output should be checked at every possible input value in the range.

- **3* faults:** the normal value of the 3 logic level is 60uA. 3* faults cause this value to be higher than 60uA. Typical examples found were in the range of 80 up to 300uA.

Test generation for these faults (in all considered circuits) will be considered after fault characterization is made. It should also be noted that all simulations will be conducted on 4-valued circuits.

4 Fault Characterization of the MVL Set

The literal operation will be selected as an example to illustrate the characterization steps conducted. Similar work has been carried out for the other four circuits. Figure 2 shows the literal implementation, found in [5], with the transistors numbered as suggested before.

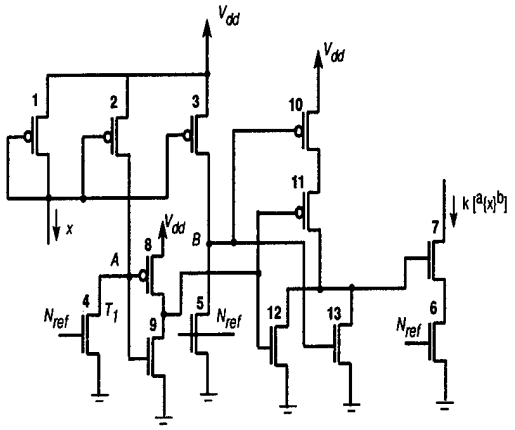


Figure 2. Literal transistor diagram.

Defects (shorts and opens) will be inserted one at a time. The circuit will then be analysed to find out the resultant behavior in the presence of the fault. This step will be repeated exhaustively for all defects (3 shorts and 3 opens per single transistor) to find resultant fault categories and their statistics.

Example 1: When the short S1gs is applied to the literal circuit, it produced a stuck at 0 (SA0) fault for all inputs. The same simulation is repeated for all possible configurations of the literal, namely $k^1[x]^1$, $k^1[x]^2$, $k^1[x]^3$, $k^2[x]^2$, $k^2[x]^3$, $k^3[x]^3$ for all possible values of k (1, 2 and 3). The fault produced the same effect, SA0, for all combinations and thus will be classified in the SA0 fault category.

Table 1 summarizes the results obtained and the fault categories identified for the literal circuit. Simulation showed that the highest percentage of faults, 40.3%, resulted in stuck at 0 output (SA0). Sequential behavior was observed in 33.8% of the cases. Functional faults, 19.5%, change the circuit's behavior into a different function from what it was designed for. The aSA0 is an example of this category.

Fault Category	Faults	Count	Percentage
Sequential	O1g, O2g, O3g, O4g, O4s, O4d, O5g, O7g, O8g, O8s, O8d, O9g, O9s, O9d, O10g, O10s, O10d, O11g, O11s, O11d, O12g, O12s, O12d, O13g, O13s, O13d	26	33.8%
SA0	S1gs, S1sd, S2gs, S3gs, S3gd, S3sd, S4gs, S4gd, S4sd, S5gs, S6gs, S7gs, S8sd, S8gd, S9gs, S9gd, S10gs, S10gd, S11gs, S12sd, S13sd, O1s, O1d, O2s, O2d, O5s, O5d, O6s, O6d, O7s, O7d	31	40.3%
SAk	S7sd	1	1.3%
Functional aSA0 bSA3 bSA3 + para_k Comp. of literal	 S2sd, S2gd, S8gs, S9sd, S11sd, S12gs, S11gd, S12gd S5sd, O3s, O3d, S13gs S13gd, S10sd S7gd	15	19.5%
Others 3* faults	 S5gd, S6gd, S6sd(kSA3*), O6G(sequential*)	4	5.2%
		total 77	

Table 1. Fault characterization of the literal circuit.

The same procedure was used to characterize faults for the other circuits in the set, namely: the Complement of literal, the Cycle, the tSum and the Min.

The cycle characterization results are shown in Table 2.

Fault Category	Faults	Count	Percentage
Functional	S2sd, S2gd, S3gd, S4gs, S4sd, S4gd, S5gs, S5gd, S5sd, S6gs, S6gd, S6sd, S7gs, S7gd, S7sd, S8gs, S8gd, S8sd, S11gs, S11gd, S11sd, S12gs, S12gd, S12sd, S13gs, S13gd S13sd, S14gs, S14gd, S14sd, O2s, O2d, O3s, O3d, O5s, O5d, O6s, O6d, O7s, O7d, O8s, O8d, O9s, O9d	44	53.7%
Sequential	O1g, O1s, O1d, O2g, O3g, O4g, O4s, O4d, O5g, O6g, O7g, O8g, O9g, O10g, O11g, O11s, O11d, O12g, O12s, O12d, O13g, O13s, O13d, O14g, O14s, O14d	26	31.7%
SA3 SA3 SA3*	 S1gs, S1sd, S2gs, S3gs S3sd, S10gd, S10sd	7	8.5%
SA0	S9gs, S9sd, S10gs, O10s, O10d	5	6.1%
		Total 82	

Table 2. Fault characterization of the cycle circuit.

The fault characterization conducted on the MVL set in this work resulted in 4 main fault categories:

1. **Sequential faults** which have sequential behavior.
2. **SA0:** the output is stuck-at zero level.
3. **SA3:** the output is stuck-at level 3. In the case of literals, this appeared as stuck at the literal value (the value of k).
4. **Functional faults** which change the circuit function into a different one from what it was designed for.

In addition, some other peculiar faults were found with low percentages. The percentages of each category against the circuits studied are tabulated in Tablestat.

	Literal	C Literal	Cycle	tSum	Min
Sequential	33.8%	33.8%	31.7%	24.6%	28.2%
Functional	20.8%	20.8%	53.7%	37.7%	30.8%
SA0	40.3%	6.5%	6.1%	7.5%	33.3%
SA3	1.3%	35.1%	8.5%	26.4%	7.7%
Others	3.9%	3.9%		3.8%	

Table 3. Overall summery.

5 Testing of the MVL Set

For a given circuit under test, the aim of any test generation process is to find the proper inputs that have their fault free outputs different from the faulty output(s). For example the normal output for $x^{\leftarrow 1}$, when x = 0, is 3. To test for SA0 fault at the output, 0 is a valid test because if it is applied, it will produce a fault-free output of 3 which is different from the faulty output (0). On the other hand, x = 1 is not a valid test because if it is applied, it will produce a fault-free output of 0 which is not distinguishable from the faulty output. Sequential faults need two test vectors since the circuit output under such faults depends on both the previous state and the current input. One vector is needed to initialize the circuit and the second vector to excite the fault.

For each of the circuits studied, test vectors were generated by simulation of each fault against all possible input variations (a sequence of two different inputs for sequential faults and single input for other faults). For the circuits considered, lists of the valid tests for every circuit under all possible faults (shorts and opens) were generated by exhaustive simulation. This is possible since the primitives are relatively small (maximum 14 transistors per circuit). From these test lists, minimal sets or input sequences that test for all faults are extracted.

The Literal tests are selected for illustration. Table 4 lists the faults and their tests for the literal $k^1[x]^3$. For each single fault (or two equivalent faults), there is one corresponding entry. Valid test vectors are listed along the row. These tests are obtained and verified by simulation. Sequential faults have their tests listed as pairs of the initializing input value and the second value as (x1, x2). x1 is the first input which should be followed by x2 to excite the fault. Functional and other non-sequential faults need only one input for test. Valid tests are listed separated by commas. Entries marked with (correct output) are for faults that do not affect the output and for which the circuit behavior is the same as that of the fault-free circuit. $Para_k$ entries require the circuit to be tested in the whole specified range. For example, to test for the short S5gd, 1, 2 and 3 are required. 3*, or k*, indicate that 3* testing procedure have to be applied in addition to the listed vectors. 3* testing will be dealt with at the end of this section.

In order to find the minimal test set(s)/sequences, we have applied the concept of fault coverage table. For example, the sets obtained for $k^1[x]^3$ are:

Fault	Tests	Fault	Tests	Fault	Tests
O1g	(0, 1) (0, 2) (0, 3) (1, 2) (1, 3) (2, 1) (2, 3) (3, 1) (3, 2)	O2g	(0, 1) (0, 2) (0, 3) (1, 0) (2, 0) (3, 0)	O3g	correct
O4s & O4d	(1, 0) (2, 0) (3, 0)	O4g	correct	O5g	correct
O7g	(1, 0) (2, 0) (3, 0) (0, 1) (0, 2) (0, 3)	O8s & O8d	(1, 0) (2, 0) (3, 0)	O8g	(0, 1) (0, 2) (0, 3) (1, 0) (2, 0) (3, 0)
O9g	(0, 1) (0, 2) (0, 3)	O10s & O10d	(0, 1) (0, 2) (0, 3)	O10g	correct
O11s & O11d	(0, 1) (0, 2) (0, 3)	O11g	(0, 1) (0, 2) (0, 3)	O12s & O12d	(1, 0) (2, 0) (3. 0)
O12g	(0, 1) (0, 2) (0, 3) (1, 0) (2, 0) (3, 0)	O13s & O13d	correct	O13g	correct
O6g (k*)	(0, 1) (0, 2) (0, 3)	SA0	1, 2, 3	SAk	0
aSA0	0	bSA3	correct	S5gd	Para_k at: all 1, 2 & 3
S13gd& S10sd	Para_k at:	S6sd	k* at 1, 2 or 3		

Table 4. $k^1[x]^3$ Literal Tests.

{(0, 3), (1, 0), 2} or {(1, 0), (0, 2), 3} or {(2, 0), (0, 1), 3} or {(2, 0), (0, 3), 1} or {(3, 0), (0, 1), 2} or {(3, 0), (0, 2), 1}

Using these sets, the minimal test sequence(s) can be found as shown in Table 5. Listed below (Table 5), are the minimal test sequences for the 6 literal instances.

Literal	Test Sequence
$k^1[x]^1$	$1 \rightarrow 0 \rightarrow 1 \rightarrow 2 \rightarrow 1$ or $1 \rightarrow 0 \rightarrow 1 \rightarrow 3 \rightarrow 1$
$k^1[x]^2$	$1 \rightarrow 0 \rightarrow 1 \rightarrow 3 \rightarrow 1$ or $2 \rightarrow 0 \rightarrow 2 \rightarrow 3 \rightarrow 2$
$k^1[x]^3$	$1 \rightarrow 0 \rightarrow 3 \rightarrow 2$ or $1 \rightarrow 0 \rightarrow 2 \rightarrow 3$ or $2 \rightarrow 0 \rightarrow 1 \rightarrow 3$ or $2 \rightarrow 0 \rightarrow 3 \rightarrow 1$ or $3 \rightarrow 0 \rightarrow 1 \rightarrow 2$ or $3 \rightarrow 0 \rightarrow 2 \rightarrow 1$
$k^2[x]^2$	$2 \rightarrow 3 \rightarrow 2 \rightarrow 0 \rightarrow 2$ or $2 \rightarrow 3 \rightarrow 2 \rightarrow 1 \rightarrow 2$
$k^2[x]^3$	$3 \rightarrow 1 \rightarrow 2$ or $2 \rightarrow 1 \rightarrow 3$
$k^3[x]^3$	$2 \rightarrow 3 \rightarrow 2$

Table 5. Minimal test sets/sequences for six literals.

Similar steps were followed to obtain the minimal test sets for the other MVL circuits. Table 6 shows those for the cycle. Due to space limitations, the test sequences for the *tsum* and the *min* are omitted.

Cycle	Test sequence
$x^{\leftarrow 1}$	$2 \rightarrow 0 \rightarrow 3$
$x^{\leftarrow 2}$	$1 \rightarrow 3 \rightarrow 0$ or $2 \rightarrow 0 \rightarrow 3$
$x^{\leftarrow 3}$	$2 \rightarrow 3 \rightarrow 1$ or $2 \rightarrow 3 \rightarrow 0$

Table 6. Minimal Test Sets/Sequences for the cycle.

5.1 Peculiar Faults Testing

Testing for 3* Faults

The 3* faults refer to any fault that results in current levels higher than 60uA. Typical values found in the simulations range from 80uA to 300uA.

If a 3* value is produced at the output of a faulty circuit, it may act as an input to another circuit. If the fault free value

for a 3* fault is 0, 1 or 2, then the fault will be detected. However, when the fault free value is 3, then the behavior of each circuit needs to be studied to find out whether this fault will produce an observable effect on the output or not. The results of this study are summarized below.

1. Literal, complement of literal and tSum gates will read the 3* value as 3 and their outputs will not be affected. This is due to the current threshold elements at the inputs of these circuits which read any value $\geq$ 60uA as 3.

2. The cycle will always produce different outputs for 3 and 3* inputs. Therefore, the fault can be detected.

3. The min will respond in two different ways. First, if one input is 3* and the other input $\neq$ 0, it will read 3* as if it was normal 3. On the other hand, if the input was $< 3^*, 0 >$ (or $< 0, 3^* >$) the output will be a recognized logic level higher than 1.

In the previously generated test sequences, the following should be note:

1. If a 3* fault is detectable, the sequence will generate a test for it. For example, SA3* fault in the cycle can be detected by tests that produce a different output from 3.

2. If the 3* fault can not be detected, the test is designed to excite the fault. For example, the sum* fault category, in the tsum, produce un-truncated sum. For inputs that have a sum $\leq$ 3, the circuit behaves like the fault-free circuit. For higher inputs, the 3* output will be produced. So, the tests are designed to **excite** the 3* output.

The circuits that produce un-detectable 3* faults are: literal, complement of literal, and tSum. For the literals, if the k is 1 or 2 and a 3* output is produced, it will be detected since it will produce different logic level, 3. For literals when k is set to 3 and the tSum, special propagation procedure needs to be taken when the correct output is 3 and the faulty output is 3* (this happens when x in [a, b] for the literal, x outside [a, b] for the complement of literal, and $x_1 + x_2 \geq 3$ for the tSum). The recommended procedure is shown below.

1. If the subsequent circuit, whose input is connected to the 3*, is literal, complement of literal or tSum, it will not be affected and will produce correct values. In this case, the error will be redundant.

2. If the subsequent circuit is a cycle, it will always produce faulty output which can then be propagated to an observable output.

3. If the subsequent circuit is a min, then the test algorithm should propagate the expected 3* twice. Once with the other min's input set to 0 and once when it is set to 3. The 0 will detect a 3* fault and the 3 will detect other faults.

The fault coverage figures for the generated test sequences in circuit are tabulated in Table 7. Two figures are given. The first assumes that the above procedure of 3* testing is applicable. The second is when the 3* procedure is not applied.

Circuit	Un-covered faults	Coverage, 3* test applicable	# of un-detectable 3* faults	Coverage, 3* test not applicable
Literal	S6gd (Osc.)	98.7%	2 (S6sd, O6g)	98.7 - (2.6 * α) 98.7% if α = 0 96.1% if α = 1
Comp. of literal	S6gd (Osc.)	98.7%	2 (S6sd, O6g)	98.7 - (2.6 * α) 98.7% if α = 0 96.1% if α = 1
Cycle	0	100%	0	100%
tSum	S6gd (Osc.) S3gd (Para.)	96.2%	11 (sum* + sequential*)	75.4%
Min	0	100%	0	100%

Table 7. Fault coverage figures and 3* testing.

α, in Table 7 is the ratio of the number of literal/complement of literal gates that are designed with k = 3 compared to the number of literal/complement of literal gates in the whole network.

Oscillations Fault

Under this fault, the circuit's output oscillates over the whole range from 0 to 60uA with high frequency. It is caused by the a continuous charging/discharging paths affecting the gate voltage (N_{ref}) of transistors 4, 5 and 6. The charging path sources from the output current (when the switch transistor 7 is open) through the short to the gate of transistor 6 (which is connected to the gates of 4 and 5). The discharging path (when transistor 7 is closed) passes through the short to the grounded source of transistor 6. This fault is not covered by the test sequences derived earlier. However, only one short produced this type of fault (S6gd in the literal and complement of literal, and S3gd in the tSum) which is a low percentage (1.3% for the literals and 1.9% for the tSum).

This can be solved for the Literal and Complement of Literal if the positions of transistors 6 and 7 (Figure 2) are swapped (a Design for Testability (DFT) issue). In this case, the fault will be converted to a normal fault that can be detected by the test sequences. This was not possible for the tSum.

A Soft Error

One short, S6gd in the tSum, resulted in a slightly different output from the fault free circuit. It only changes the current value of logic 3 (normally 60uA) into an intermediate level (50uA) between 2 and 3. This can be read as 3, and passes undetected, or as 2 which can be detected. This can not be determined and is considered uncovered by the test

sequences produced.

6 Recommendations for Testability of MVL Circuits

Based on the results obtained in this work, the following general recommendations can be made for the set of MVL circuits considered:

1. Literals and tSum should not be used as output stages. This is to avoid 3* values at the output where its effect can propagate to another circuit. Usually, literals appear at the input stages and do not appear afterwards.

2. If a tSum has to be used at the output, it should be followed by 2 consecutive cycle gates. This will guarantee the detection of the 3* faults if they appear at the tSum output.

3. Apply 3* test procedure for every min preceded by literals or tSum. This is to properly propagate possible 3* values generated by the literals or the tSum.

4. For the Literal and Complement of Literal gates, swap the positions of transistors 6 and 7. This will eliminate the oscillation fault caused by the S6gd short and will convert it to a testable fault.

7 Concluding Remarks and Future Work

In this paper, fault characterization and testability considerations in a set of MVL basic gates were considered. To achieve this, faults (shorts and opens) were inserted one at a time and each circuit was simulated for all possible input transitions to find out which inputs are valid to test for the fault inserted. The process was repeated for each circuit for all its faults and the resultant test vectors for all faults were tabulated. Then these tables were used to generate minimal test sequences for each circuit.

The main conclusions that can be derived from this work are:

1. The stuck-at fault model is inadequate for representing faults in the MVL circuits. Actually, it represented less than 40% of the total faults.

2. High percentage of faults appeared as functional faults, which change the circuit output into different function. They ranged from 20% in the literals to more than 50% in the cycle.

3. Only SA0 or SA3 faults appeared. There are no stuck-at faults at the other logic levels 1 or 2 (stuck-at the literal value, in the literals, is a special case of SA3).

After using the testability recommendations in this work, the given test sequences can be used in testing larger circuits. The authors are currently working on the issue of precomputed test sets for circuits using the considered MVL set as basic building blocks.

Acknowledgment

The authors would like to acknowledge KFUPM for its continued support.

References

[1] Rob Dekker, Frans Beenker, and Loek Thijssen. A Realistic Fault Model and Test Algorithms for Static Random Access Memories. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 9(6):567–572, 1990.

[2] Hong Hao and Edward J. McCluskey. Analysis of Gate Oxide Shorts in CMOS Circuits. *IEEE Transactions on Computers*, 42(12):1510–1516, December 1993.

[3] S. Hessabi, M. Y. Osman, and M. I. Elmasry. Differential BiCMOS Logic Circuits: Fault Characterization and Design-for-Testability. *IEEE Transactions on VLSI Systems*, 3(3):437–445, September 1995.

[4] Marcel Jacomet and Walter Guggenbuhl. Layout-Dependent Fault Analysis and Test Synthesis for CMOS Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(6):888–899, 1993.

[5] Atul K. Jain, Ron J. Bolton, and Mostafa H. Abd-El-Barr. CMOS Multiple-Valued Logic Desgin-Part I: Circuit Implementation. *IEEE Transactions on Circuits and Systems-I:Fundamental Theory and Applications*, 40(8):503–514, 1993.

[6] Atul K. Jain, Ron J. Bolton, and Mostafa H. Abd-El-Barr. CMOS Multiple-Valued Logic Desgin-Part II: Function Realization. *IEEE Transactions on Circuits and Systems-I:Fundamental Theory and Applications*, 40(8):515–522, 1993.

[7] Siyad C. Ma and Edward J. McClulskey. Non-Conventional Faults in BiCMOS Digital Circuits. In *International Test Conference*, pages 882–891, 1992.

[8] Mohamed Y. Osman and Mohamed I. Elmasry. Highly Testable Design of BiCMOS Logic Circuits. *IEEE Journal of Solid-State Circuits*, 29(6):671–678, 1994.

[9] B.E. Stewart, D. Al-Khalili, and C. Rozon. Defect Modelling and Testability Analysis of BiCMOS Circuits. *Can. J. Elec. & Comp. Eng.*, 16(4):148–153, 1991.

HIGHLY TESTABLE BOOLEAN RING LOGIC CIRCUITS

Ugur Kalay, Marek A. Perkowski, Douglas V. Hall

Dept. of Electrical and Computer Engineering
Portland State University
P.O. Box 751, Portland, OR 97207-0751
ugurkal@ee.pdx.edu, mperkows@ee.pdx.edu, dough@ee.pdx.edu

Abstract

In this paper we show how Boolean Ring logic, a group-based logic, leads to a circuit implementation that is highly testable. We develop Boolean Ring based expressions, which we call Generalized-Literal Boolean-Ring Sum-of- Products (GL-BRSOP) and Universal-Literal Boolean-Ring Sum-of-Products (UL-BRSOP) to represent (powers of 2)-valued MVL functions. Our BRSOPs: allow MVL functions to be implemented with vectors of binary AND and EXOR gates; allow the use of a binary logic fault model; and allow a time-multiplexed implementation that is highly testable and uses less redundant circuitry.

1. INTRODUCTION

Boolean Ring is a group-based logic system like Galois Fields. A Galois Field, GF(k^r), is a set where k is a prime number and r is a positive integer [3, 13]. AND-EXOR logic systems correspond to the 2-valued Galois Field, GF(2). It is well known that binary AND-EXOR circuits are easily testable. Researchers have introduced highly testable implementations and test sets for various AND-EXOR forms such as PPRM, FPRM, GRM, and ESOP [16, 17, 19, 20]. The testability of multiple-valued Galois Field logic circuits has also been investigated. Dubrova investigated GF(k) circuits [6], where k is a prime number; and in [11], we proposed a highly testable realization and a testing scheme for the most general functional representation, Galois Field Sum-of- Products (GFSOP), over GF(k^r), where r is equal to or greater than 1.

+	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

·	0	1	2	3
0				
1		1	2	3
2		2	3	1
3		3	1	2

Figure 1. GF(4) addition and multiplication tables.

The high testability of GF circuits is due to the fact that the GF operators exhibit the cyclic group property. This property can be explained from the 4- valued (quaternary) GF operators as shown in Figure 1. Note that in any row of the addition table in Figure 1, the elements are all different, which is cyclic, and that the elements have a different order in each row. Another cyclic group can be observed in the multiplication table. If the zero element (shaded in gray) is removed from the multiplication table, then the remaining elements form a cyclic group. In binary, the GF(2)-addition operator, EXOR, has the cyclic group property. This characteristic is useful in detecting faults because any single change in the inputs of the gate will cause a change in the output.

Our work with GF(k^r) circuits led us to investigate other logic systems that exhibit the cyclic group property. One such logic system is Min-ModSum, which was invented by Dueck and Miller [7]. The operators of Min-ModSum, Min and ModSum, perform the Minimum operation and Modulo-Summation, respectively. The testability of Min-ModSum circuits is similar to that of multiple-valued GF circuits because ModSum forms a cyclic group with all the elements of the logic. However, Min does not have any cyclic group, so the test set is not superior to that of GFSOPs given in [11]. Remember that GF-multiplication has a cyclic group when the zero element is excluded.

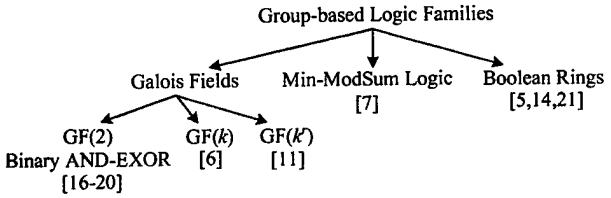


Figure 2. The group-based logic families.

The contribution of this paper is the analysis of Boolean Ring [5, 14, 21], BR(2^r), as another group-based multiple-valued logic with 2^r elements, where r is a positive integer number. Figure 2 summarizes the currently investigated group-based logic systems. One major advantage of BR is that it allows MVL functions to be implemented in space with vectors of binary AND and EXOR gates. As we show later, when implemented in time, BR also leads to highly testable circuits with efficient test sets. This fact was not realized by previous researchers. The implementation in time is realized by using registers and by time-multiplexing the circuit. Our work has significant practical implications because BR circuits can be implemented with very fast opti-

0-7695-0161-3/99 $10.00 © 1999 IEEE

cal realizations [9]. Also, some recent patents emphasize a time-multiplexed logic that can be used to implement BR functions realized in FPGAs [12, 22].

The organization of this paper is as follows. In Section 2, we introduce BR functional representations and demonstrate the functional completeness of these forms. In Section 3, we show highly testable implementations of these forms and give an efficient testing scheme. In Section 4, we give our conclusions and plans for future work.

2. BR(2^r) FUNCTIONAL REPRESENTATIONS

2.1. Binary Vector Interpretation of Boolean Ring

In this section, we will define a new interpretation that uses binary gates to implement arbitrary, 2^r-valued functions. In this paper, we assume that each MVL variable is represented by r binary wires, or rather, an ordered binary vector with r elements. This scheme allows us to implement the circuit with vectors of binary gates and to simply increase the size of the vector linearly as the radix is increased. As an example, Table 1 shows the representation (encoding) for quaternary logic, the case where $r = 2$.

00	0
01	1
10	2
11	3

Table 1. The binary encoding for quaternary logic levels.

Most multiple-valued systems have two operators: *addition-like* and *multiplication-like*. The multiplication-like operator is used to cut a portion(s) of the entire function, and the addition-like operator is used to combine these sub-functions to create the complete function. Therefore, in any complete logic system, a function can be represented in Sum-of-Products form, or rather, the combination of sub-functions. In GF, for example, the GF-multiplication implements the cutting function, and the GF-addition implements the combining function. (Hozumi, et al., have shown the existence of other choices to implement the cutting and combining functions in [10].) Since our objective was to have a highly testable implementation, we felt it very important to select operators that have some form of a cyclic group.

It can be shown from the axioms of Boolean Ring that a vector of r, binary EXOR gates can be used to implement the BR-addition operation with our vector approach and the encoding in Table 1. As explained earlier, the binary EXOR gate exhibits the cyclic group property and thus leads to easily testable circuit implementations. Figure 3 shows the vector of EXOR gates for a quaternary logic function with multiple-valued input variables A and B. Note that each input variable and the output are represented by two wires. For MVL with $2^{r=3} = 8$ values, three EXOR gates would be required for the vector.

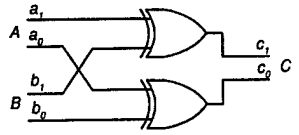


Figure 3. The vector of EXORs that implements the BR-addition in quaternary logic.

Definition 1: "$\oplus\!\!\oplus$" represents the function realized by a vector of EXORs, which we will call the BR-addition operator. (The two EXOR symbols imply the vector, not the number of EXOR gates used in the realization of the operator.)

Example: BR-addition of two quaternary constants: $2\oplus\!\!\oplus 3 = [10]_{binary}\oplus\!\!\oplus[11]_{binary} = [01]_{binary} = 1$.

Figure 4 shows the truth table for the vector of EXORs, both in binary and MVL form. By comparison with Figure 1, it can be seen that this circuit implements the 4-valued GF-addition.

	00	01	10	11
00	00	01	10	11
01	01	00	11	10
10	10	11	00	01
11	10	01	00	

c_1c_0

=>

	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

C

Figure 4. The truth table for the vector of EXORs and the corresponding quaternary table.

To implement the cutting operation, let's investigate the properties of a vector of ANDs such as that shown in Figure 5. The truth table in Figure 6 shows that our vector AND circuit does not implement the GF-multiplication operator. Furthermore, it does not exhibit the cyclic group property. However, similar to the Min operator in Min-ModSum logic, it has the **0** (zero) element and the **1** (multiplicative-identity) element of a logic system. In quaternary, these are '0' for the **0** element, and '3' for the **1** element.

Definition 2: "$\bullet\!\!\bullet$"represents the function realized by a vector of ANDs, which we will call the BR-multiplication operator.

Example: BR multiplication of two quaternary constants: $2\bullet\!\!\bullet 3 = [10]_{binary}\bullet\!\!\bullet[11]_{binary} = [10]_{binary} = 2$.

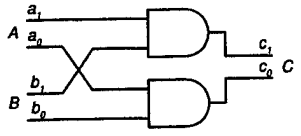


Figure 5. The vector of ANDs.

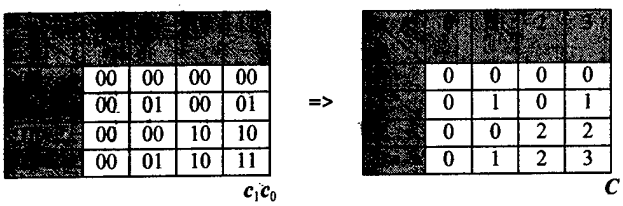


Figure 6. The truth table for the vector of ANDs and the corresponding quaternary table.

2.2. Boolean Ring Sum-of-Products Forms

Definition 3: A *literal* is a one-variable function that has the minimum or the maximum value of the logic system as its value. It has the maximum value for the input values specified as the left superscript, and the minimum value for the remaining input values.

Definition 4: If a literal is in a simple form, and has only one input value specified in its left superscript, then the literal is called a *Post literal* or the *characteristic function* of a variable. For example, in ternary logic, a Post literal ${}^0x$ takes the value '2' for $x = 0$, and '0' for the remaining input values ('1' and '2').

Definition 5: If a literal has more than one input value in its left superscript, then the literal is called a *generalized literal*. For example, in quaternary logic, a generalized literal, ${}^{1,2}x$ takes the value '3' for $x = 1$ and $x = 2$, and takes the value '0' for the other input values, $x = 0$ and $x = 3$. Note that the Post literal is a special case of the generalized literal.

Definition 6: A more complex literal, *universal literal*, is defined by Fraser and Dueck in [8] as follows. Let x_i be a variable that can assume any of the logic values in the set $\{0, 1, 2, \ldots, (N\text{-}1)\}$, where N is the radix. Let $\boldsymbol{X} = \{x_1, x_2, \ldots, x_n\}$ be a set of n variables. A universal literal is a one-variable function $<a_0a_1 \ldots a_{N-1}>x_i$, where $<a_0a_1 \ldots a_{N-1}>x_i = a_{x_i}$. For example, in quaternary logic, a universal literal $<1003>A$ of a quaternary variable A takes the value '1' for $A = 0$; takes the value '0' for $A = 1$ and $A = 2$; and takes the value '3' for $A = 3$ according to the definition.

The relation of complexity among the different types of literals in our context can be illustrated as in Figure 7. The universal literals have the most complexity among the three.

Assuming x is an MVL variable, the following axioms can be observed for the BR multiplication operator:

1. $x \bullet\bullet \mathbf{0} = \mathbf{0} \Rightarrow x \bullet\bullet 0 = 0$ (in quaternary)
2. $x \bullet\bullet \mathbf{1} = x \Rightarrow x \bullet\bullet 3 = x$ (in quaternary)

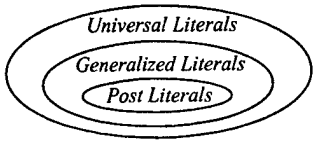


Figure 7. The inclusion of the complexity of the literals.

These properties allow us to construct our sub-functions (product terms) based on literals, rather than variables or powers of variables as in GF product terms. The practical result is that BR logic functions can be expressed with just constants, literals and BR vector operators.

Definition 7: A Boolean Ring form, *Generalized-Literal Boolean-Ring Sum-of-Products* (GL-BRSOP), can be defined as:

$f = (c_1 \bullet\bullet x_{11} \bullet\bullet x_{12} \bullet\bullet \cdots \bullet\bullet x_{1n}) \oplus\oplus (c_2 \bullet\bullet x_{21} \bullet\bullet x_{22} \bullet\bullet \cdots \bullet\bullet x_{2n}) \oplus\oplus \cdots \oplus\oplus (c_p \bullet\bullet x_{p1} \bullet\bullet x_{p2} \bullet\bullet \cdots \bullet\bullet x_{pn})$,

where it is assumed that there are n variables and p product terms; c_i is a constant such that $1 \leq c_i \leq (2^r\text{-}1)$; x_{ij} is a generalized literal of the j^{th} variable in the i^{th} product term, and it can be a different literal of the same variable at each product term. For a quaternary variable, x_{ij}, can be ${}^0x$, ${}^{0,1}x$, ${}^{2,3}x$, etc. The function, $f = (2 \bullet\bullet {}^1x_1 \bullet\bullet {}^{0,3}x_2) \oplus\oplus (1 \bullet\bullet {}^{1,2,3}x_1 \bullet\bullet {}^2x_2) \oplus\oplus (2 \bullet\bullet {}^{0,2}x_1 \bullet\bullet {}^3x_2)$, is an example of a GL-BRSOP expression. Notice that the expression can contain both Post literals and generalized literals. If only Post type literals are used in the product terms (minterms), the GL- BRSOP expression is a canonical form analogous to the binary Minterm Canonical Form. Otherwise, the expression is non-canonical analogous to the binary SOP expression. Our next step is to prove the generality of our formulation.

Theorem 1: Every multiple-valued logic function with 2^r logic values can be realized in Generalized-Literal Boolean-Ring Sum-of-Products (GL-BRSOP) form.

Proof: Like binary SOP expressions, a GL-BRSOP expression can be written by BR-adding the minterms that are constructed for each entry in the truth table where the function is not zero. In such case, the BRSOP expression is non minimal, and based on Post literals. A minterms is constructed such that it gets the value of the function in the corresponding entry. The minterm construction is done by BR-multiplying a constant and all the Post literals constructed for the entry. Since each Post literal in the minterm gets the value of the **1** element for the entry, the multiplication of them gives the **1** element. Therefore, the constant is chosen such that the minterm gets the value of the function for that entry. As a result, the BR-addition of all the minterms constructed in this manner realizes the entire function because the expression gets the value of the minterm for the corresponding entry where all other minterms get the value of the zero element. □

Example: We will use the quaternary MVL function whose truth table is shown in Table 2 to illustrate the construction of a GL-BRSOP expression.

For each entry that does not have a '0' output, we write a Post literal based minterm using BR-multiplication and then combine these minterms using BR-addition to generate the BRSOP expression. For the first entry in the truth table, the minterm is $M_1 = (2 \bullet\bullet {}^0A \bullet\bullet {}^1B)$. Without the constant coefficient, the minterm value is the **1** element (=3)

0 1	2
1 2	2
3 1	3
3 3	1
Others	0

Table 2. A quaternary MVL function.

for the input values in the first entry because each literal in the minterm gets the value of the **1** element ($^0A \bullet\bullet {}^1B = 3 \bullet\bullet 3 = 3$). However, we want the value of the minterm to be '2' to implement the example function for this entry. Therefore, the multiplication with constant '2' gives the correct output ($2 \bullet\bullet 3 = 2$). Obviously, this correction is not necessary for the minterms where the desired result is '3' (the **1** element). After obtaining all the minterms in the same manner, the GL-BRSOP expression that represents the entire function is: $F = M_1 \oplus\oplus M_2 \oplus\oplus M_3 \oplus\oplus M_4 = (2 \bullet\bullet {}^0A \bullet\bullet {}^1B) \oplus\oplus (2 \bullet\bullet {}^1A \bullet\bullet {}^2B) \oplus\oplus ({}^3A \bullet\bullet {}^1B) \oplus\oplus (1 \bullet\bullet {}^3A \bullet\bullet {}^3B)$.

Definition 8: A Boolean Ring non-canonical expression, *Universal-Literal Boolean-Ring Sum-of-Products* (UL-BRSOP), can be defined as:

$f = (c_1 \bullet\bullet x_{11} \bullet\bullet x_{12} \bullet\bullet \cdots \bullet\bullet x_{1n}) \oplus\oplus (c_2 \bullet\bullet x_{21} \bullet\bullet x_{22} \bullet\bullet \cdots \bullet\bullet x_{2n}) \oplus\oplus \cdots \oplus\oplus (c_p \bullet\bullet x_{p1} \bullet\bullet x_{p2} \bullet\bullet \cdots \bullet\bullet x_{pn})$,

where it is assumed that there are n variables and p product terms; c_i is a constant such that $1 \leq c_i \leq (2^r - 1)$; x_{ij} is a universal literal of the j^{th} variable in the i^{th} product term, and it can be a different literal of the same variable at each product term. For example $f = (<2100>x_1 \bullet\bullet <0331>x_2) \oplus\oplus (<0330>x_1 \bullet\bullet <1123>x_2)$ is a quaternary UL-BRSOP expression. Notice that in contrast to GL-BRSOP, the expression does not contain constant coefficients because a universal literal can have any value.

UL-BRSOP expressions are also functionally complete since they include GL-BRSOP expressions. We will give the following example to illustrate how universal literals can be used to express an MVL function in BR.

Example: Let's find a UL-BRSOP expression for the 2-variable quaternary function below:

0	0	0	0	0
1	0	3	3	2
2	0	3	2	3
3	0	0	1	3
				F

Table 3. A quaternary function.

Since UL-BRSOP is not a canonical form, the solution we give below is just one of the possible solutions to the problem. Searching for the minimum solution to the problem could be done with a heuristic scheme similar to that described in [8], or the approximate/exact backtracking minimization scheme that was proposed in [23] and extended in [15]. Other algorithms such as Greedy, Genetic, Tabu-Search, and Simulated Annealing could also be adapted. Our solution for the function is: $F = (<0233>A \bullet\bullet <0013>B) \oplus\oplus (<0330>A \bullet\bullet <0330>B)$. Note that, unlike a GL-BRSOP, the solution does not contain a minterm for each entry in Table 3. The easiest way to verify this solution is by looking at the truth tables for the universal literals of the expression and performing the indicated BR vector operations on a bit-by-bit basis for each entry as shown in Figure 8. For example, if '2' ($[10]_{binary}$) and '3' ($[11]_{binary}$) are vector-ANDed ($\bullet\bullet$) together, then the result is '2' ($[10]_{binary}$).

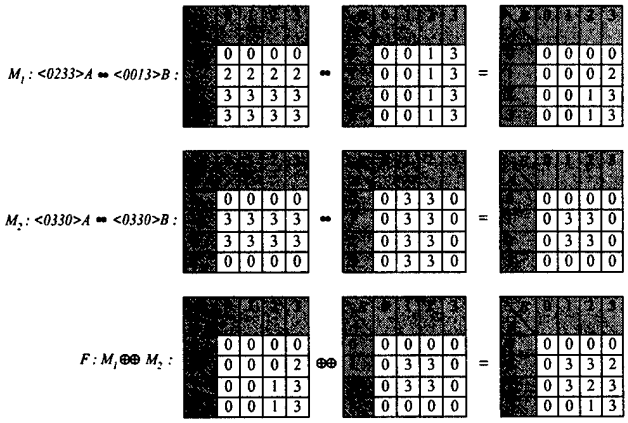


Figure 8. The functional verification of the example UL-BRSOP expression.

3. EASILY TESTABLE IMPLEMENTATIONS

3.1. The BRSOP Implementations in Space

To illustrate, we will show how our earlier GL-BRSOP example, $F = (2 \bullet\bullet {}^0A \bullet\bullet {}^1B) \oplus\oplus (2 \bullet\bullet {}^1A \bullet\bullet {}^2B) \oplus\oplus ({}^3A \bullet\bullet {}^1B) \oplus\oplus (1 \bullet\bullet {}^3A \bullet\bullet {}^3B)$ can be implemented with binary gates. Figure 9 shows a high level MVL schematic of the implementation. The BR operators, the vector of ANDs and the vector of EXORs, are symbolized using a black bar inside the conventional AND and EXOR symbols. The vector network can be implemented either as a tree, or as a cascade as shown in Figure 9. Our purpose in choosing the cascade architecture will be explained in Section 3.3.

The binary implementations for required literals can be found directly from their definitions. The truth table and circuit in Figure 10, for example, show how to obtain the literal ^0A for quaternary variable A. If only Post literals are used in the implementation, the literals can always be implemented using a single AND gate with NOT. The reason is that there

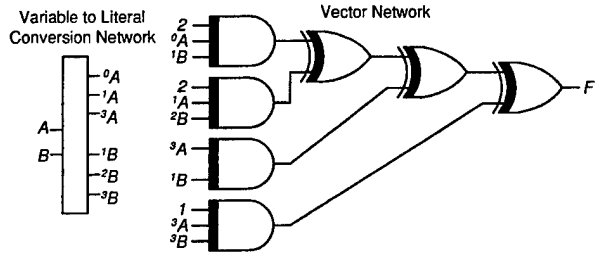


Figure 9. The MVL schematic for the BR logic implementation in space.

is only one input combination that would make the Post literal have the value of the **1** element. Using the more complex generalized literals to obtain the function may produce fewer product terms in the GL-BRSOP expression, but the binary implementation of these complex literals may require multiple gates, such as AND, OR, EXOR, XNOR, and NOT; with multiple levels. For instance, an 8-valued literal ${}^{1,6}x$ has the value of the **1** element for two input combinations: $[001]_{binary}$ and $[110]_{binary}$, which requires two AND gates in the first level and one OR gate in the second level of the binary implementation of this literal.

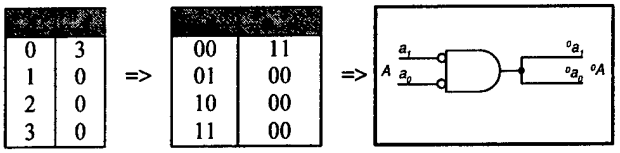


Figure 10. The binary implementation of a Post type literal.

The first minterm (M_1) of the example function can be implemented as shown in Figure 11a. The minterms produced in this manner are then combined with a vector of binary EXOR gates. Note that, as is often the case, multiplication by a constant simplifies final minterm implementation. This is illustrated in Figure 11b where the literals are multiplied by the constant '2'.

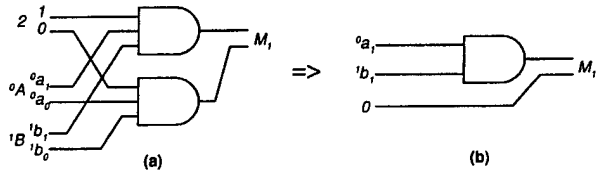


Figure 11. A product term realization before and after constant multiplication.

Figure 12 shows our complete binary implementation for the example function, F (f_1 and f_0). Notice the repetition (r times, $\log_2[radix]$) in the binary AND-EXOR network for each bit of a variable. This property simplifies routing because r-tuples of wires representing the same variable are always routed together, thus the number of connections actually routed is minimized. Furthermore, the circuit can be simplified for constant inputs before realization as shown in Figure 11b.

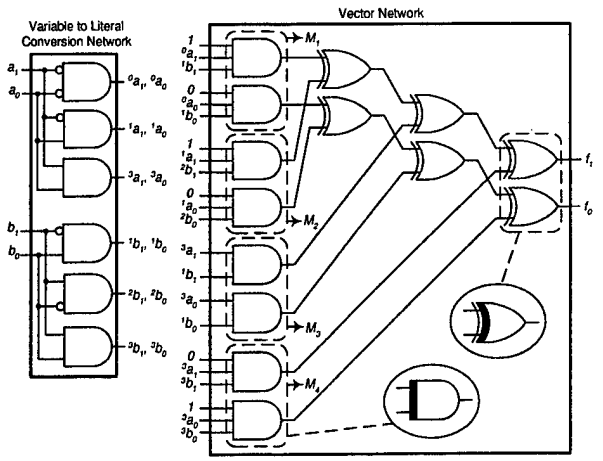


Figure 12. The quaternary example function implemented in BR logic.

As mentioned earlier, the variable-to-literal conversion network requires only single gates, if Post type literals are used. However, for complex functions, the number of gates required to implement the function using such literals may become excessive since this kind of circuit requires creating a minterm for each entry in the truth table. This problem can be overcome by using more complex generalized literals. More complex generalized literals reduce the number of product terms at the cost of increased complexity in the Variable-to-Literal conversion circuitry, but in most cases, it leads to a general reduction, especially for large circuits. Therefore, it is reasonable to assume synthesis with the most complex literals possible, i.e. universal literals.

Universal literals can be realized with binary gates in the same manner as the generalized literals explained earlier. Figure 13 shows the implementation of the universal literal $<0221>A$. Note that the realization requires a more complex realization than would a Post literal. For higher radix values, universal literal realization may even require multiple level networks.

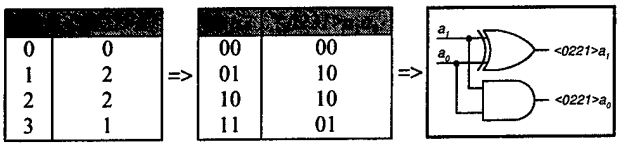


Figure 13. The binary implementation for a quaternary universal literal.

As a complete example, Figure 14 shows the implementation for our earlier UL-BRSOP expression $F = (<0233>A \bullet\bullet <0013>B) \oplus\oplus (<0330>A \bullet\bullet <0330>B)$.

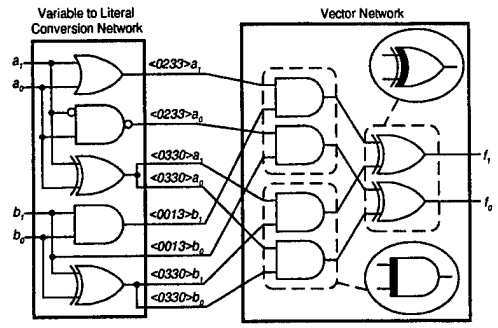


Figure 14. The UL-BRSOP implementation in space.

The operators of BR logic are somewhat similar to those of Min-ModSum logic [7], but in contrast to all other MVL systems, they can be time-multiplexed as will be explained in the following section.

3.2. The BRSOP Implementations in Time

Unlike most other multiple-valued logic systems, our interpretation of Boolean Ring logic can be time-multiplexed to improve its testability. To see the rationale for this, first note that in the BRSOP realizations shown in Figure 12 and 14, there is a repetition of r times in the vector network section. The vector network essentially implements multiple *Positive Polarity Reed Muller* (PPRM) networks. Also, note that these networks function in parallel, independently from each other. Therefore, as we will illustrate here, a single PPRM network can be time- multiplexed with the aid of shift registers, which also function as scan registers during test. Other time-multiplexing schemes [2, 12, 22] can also be utilized for this purpose. Figure 15 shows the time-multiplexed version of the circuit in Figure12. Since the PPRM network is time shared, there is only one PPRM network instead of r PPRM networks as in the vector network.

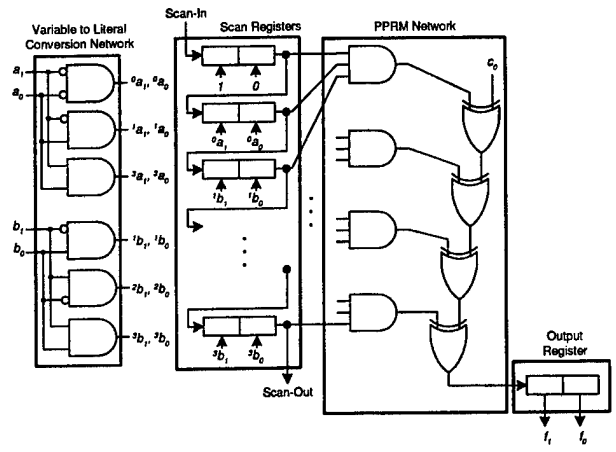


Figure 15. The binary schematic for the time-multiplexed version of the GL-BRSOP implementation.

The scan registers have two distinct modes of operation: normal mode and test mode. During the normal mode of operation, the literal values coming from the conversion part are loaded into the registers in parallel on the first clock pulse. The literal bits for the binary function f_0 are then applied to the PPRM network and the result propagates to the output register. On the second clock pulse, the result for function f_0 is serially loaded into the output register and the literal bits for function f_1 are shifted into the inputs of the PPRM network. On the third clock pulse, the result for function f_1 is serially loaded into the output register. At this point all the bits for the MVL function F (f_1 and f_0) can be read in parallel from the output register.

3.3. The Testing Scheme

During the test mode, all scan registers act as a single shift register connected as a chain. The scan registers assist in testing because they isolate the conversion network from the PPRM network, enabling these parts to be tested individually. Test vectors are shifted into the scan chain from the Scan-In input to apply them to the PPRM network; and the results are collected from the primary circuit output, the output register. The test vectors are applied to the primary circuit inputs to test the conversion network, and the results are collected by the scan chain and shifted out from the Scan-Out output.

We have chosen a cascade architecture for the BR-addition gates in the vector network of our implementation as shown in Figure 9. As a result, the resulting PPRM network in Figure 15 can be tested for multiple faults with a universal test set described by Reddy and Saluja in [18]. The c_0 input is introduced in compliance with their scheme. On the other hand, there are testing schemes for tree type implementations, such as the one presented in [20]. However, such schemes may introduce extra circuitry and/or additional test vectors for testability.

The PPRM network could be replaced, for example, with an arbitrary AND-OR network. However, in this case the test set would have to be obtained with a test generation algorithm, such as D-algorithm, because such an arbitrary network is not as easily testable as a PPRM.

The conversion network is implemented as series of single gates if Post type literals are used; or implemented as series of binary networks if more complex literals are used. Therefore, for multiple faults, all of the gates (or networks) can be tested exhaustively in parallel. The number of tests for each gate (or network) in the conversion network then is 2^r, the radix. Since the gates (networks) are tested in parallel, this is also the number of tests required for the entire conversion network.

Testing the scan registers is also done separately. This is left out of this paper since we are using conventional scan registers that have a well- known testing schemes described by other researchers [1, 4].

4. CONCLUSIONS

In this paper we showed how group-based Boolean Ring logic system leads to circuit implementations that are easily testable. The simplest example of a group-based logic system is the familiar binary AND-EXOR logic. As a further exploration of group-based logic systems, we observed that Boolean Ring addition operator has the cyclic group property that provides the high testability in all group-based logic systems.

We developed functional representations over Boolean Ring for 2^r-valued functions, where r is a positive integer number. For our first form, which we call a GL-BRSOP or Generalized-Literal Boolean Ring Sum-of-Products, the literals are found directly from the truth table of the MVL function. We illustrated the functional completeness of the new form with a quaternary example. To further extend this BRSOP representation, we defined another form, which we call UL- BRSOP or Universal Literal Boolean Ring Sum-of-Products. The universal literals for this form are more complex to generate, but they reduce the number of product terms.

A very important feature of our BRSOP representations is that they can be implemented with a vector of binary AND gates and a vector of binary EXOR gates. Furthermore, we showed that this implementation in space can also be done in a multiplexed form in time. Our time-multiplexed implementation has much less duplicated circuitry and is highly testable using standard scan techniques.

In summary, the major advantages of our BRSOPs are: they allow MVL functions to be implemented with vectors of binary AND and EXOR gates; they allow the use of a binary logic fault model; and they permit a time-multiplexed implementation that is highly testable and uses less redundant circuitry. We are currently performing some measuments with benchmark circuits to compare our scheme with classical binary circuits. We also plan to do further research with our BRSOPs, and apply the proposed techniques to Min-ModSum and other group-based combinational logic circuits.

References

[1] M. Abramovici, M. A. Breuer, and A. D. Friedman. *Digital Systems Testing and Testable Design*. IEEE Press, New York, 1990.

[2] H. H. Babu and T. Sasao. "Design of Multiple-Output Networks Using Time Domain Multiplexing and Shared Multi-Terminal Multiple-Valued Decision Diagrams". In *ISMVL'98*, pages 45–51, 1998.

[3] J. R. Batisda. *Field Extensions and Galois Theory*. Cambridge University Press, New York, 1984.

[4] N. C. Berglund. "Level Sensitive Scan Design Tests Chips, Boards, Systems". *Electronics Int.*, 52(6):108–110, Mar. 1979.

[5] G. Boole. *An Investigation of the Laws of Thought*. Walton, London, 1854. Reprinted by Dover Books, New York, 1954.

[6] E. V. Dubrova and J. C. Muzio. "Testability of Generalized Multiple-Valued Reed-Muller Circuits". In *ISMVL'96*, pages 56–61, 1996.

[7] G. W. Dueck and D. M. Miller. "A 4-Valued PLA Using the MODSUM". In *ISMVL'86*, pages 232–240, 1986.

[8] B. Fraser and G. W. Dueck. "Multiple-Valued Logic Minimization Using Universal Literals and Cost Tables". In *ISMVL'98*, pages 239–244, 1998.

[9] K. Ghosh, P. P. Choudhury, and A. Basuray. "VLSI Quadruple-Valued Logic Architecture Using Optoelectronics for Computer and Communication". Private information.

[10] T. Hozumi, O. Kakusho, and Y. Hata. "On Low Cost Realization of Multiple-Valued Logic Functions". In *ISMVL'98*, pages 233–238, 1998.

[11] U. Kalay, D. V. Hall, and M. A. Perkowski. "A Minimal and Universal Test Set for Multiple-Valued Galois Field Sum-of-Products Circuits". In *7th Workshop on Post-Binary ULSI Systems*, pages 50–51, Fukuoka Software Research Park, Fukuoka, Japan, May 1998.

[12] L. A. Lev. "Time Multiplexed Ratiod Logic - US Patent 5,612,638". http://128.109.179.23/access/search-num.html.

[13] R. Lidl and H. Niederreiter. *Finite Fields - Encyclopedia of Mathematics and Its Applications*. Addison-Wesley Publishing Company, Reading, Mass., 1983.

[14] Z. Nakao. "An Extention of Logic Functions on the Quaternary Boolean Ring". *IECE Transactions*, E69(11):1169–1172, 1986.

[15] M. Perkowski, P. Lech, Y. Khateeb, R. Yazdi, and K. Regupathy. "Software-Hardware Codesign Approach to Generalized Zakrevskij Staircase Method for Exact Solutions of Arbitrary Canonical and Non-Canonical Expressions in Galois Logic". In *6th Workshop on Post-Binary ULSI Systems*, pages 41–44, Nova Scotia, Canada, May 1997.

[16] D. K. Pradhan. "Universal Test Sets for Multiple Fault Detection in AND-EXOR Arrays". *IEEE Trans. Comp.*, 27(2):181–187, Feb. 1978.

[17] S. M. Reddy. "Easily Testable Realizations for Logic Functions". *IEEE Trans. Comp.*, C-21:1183–1188, Nov. 1972.

[18] K. K. Saluja and S. M. Reddy. "Fault Detecting Test Sets for Reed-Muller Canonic Networks". *IEEE Trans. Comp.*, C-24:995–998, Oct. 1975.

[19] A. Sarabi and M. A. Perkowski. "Design for Testability Properties of AND-EXOR Networks". In *IFIP W.G. 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, pages 418–424, Hamburg, Germany, Sept. 1993.

[20] T. Sasao. "Easily Testable Realizations for Generalized Reed-Muller Expressions". *IEEE Trans. Comp.*, 46(6):709–716, June 1997.

[21] M. Stone. "The Theory of Representation for Boolean Algebra". *Trans. of American Math. Society*, 40:37–111, 1936.

[22] S. M. Trimberger, R. A. Carberry, R. A. Johnson, and J. Wong. "Time Multiplexed Programmable Logic Device - US Patent 5,646,545". http://128.109.179.23/access/search-num.html.

[23] A. D. Zakrevskij. "Minimum Polynomial Implementation of Systems of Incompletely Specified Boolean Functions". In *Reed Muller'95*, pages 250–256, 1995.

Self-Checking Multiple-Valued Circuit Based on Dual-Rail Current-Mode Differential Logic

Takahiro Hanyu, Tsukasa Ike and Michitaka Kameyama
Department of Computer and Mathematical Sciences,
Graduate School of Information Sciences,
Tohoku University
Aoba-yama 05, Sendai 980-8579, Japan
hanyu@kameyama.ecei.tohoku.ac.jp

Abstract

A multiple-valued current-mode (MVCM) circuit based on dual-rail differential logic has been proposed for high-speed arithmetic systems at a low supply voltage. This paper presents a new totally self-checking circuit based on dual-rail MVCM logic, where almost all the basic components except a differential-pair circuit have been already duplicated, which results in small hardware overhead compared with a non-self-checking circuit based on dual-rail MVCM logic. Moreover, the performance of the proposed self-checking circuit is superior to that of full duplication of the non-self-checking circuit based on dual-rail MVCM logic in terms of transistor counts, switching delay and dynamic power dissipation under a 0.5-μm standard CMOS technology

1. Introduction

Fault-tolerance techniques, as well as high-performance system architectures and circuit design techniques [1]-[3], have become increasingly important in recent submicron VLSI chips. One of the key technologies for the detection of faults inside VLSI chips is to use a self-checking circuit that has the capability to test for the occurrence of a fault within the circuit by a normal input, as well as to detect an error at the input itself. Furthermore, by the use of self-checking techniques, transient and permanent faults can be also detected during normal operations [4], [5]. However, a self-checking circuit based on the conventional binary gates requires about twice as many transistors as the corresponding non-self-checking binary circuit, which limits a wide use of the self-checking system in the present VLSI chips.

On the other hand, an MVCM circuit based on dual-rail differential logic has been proposed for high-speed arithmetic systems at a low supply voltage. A dual-rail MVCM circuit is used to make a signal-voltage swing small yet driving capability large. In fact, the use of dual-rail MVCM logic in a 1.5V-supply multiple-valued multiplier chip enables high-speed operations with reduced transistor and interconnection counts at low power dissipation in comparison with that of a corresponding binary CMOS one [6], [7]. Moreover, in the dual-rail MVCM circuit, there are three basic components, that is, an adder for current-mode linear sum, a comparator and a differential-pair circuit (DPC) to generate a dual-rail multiple-valued current output. These first two basic components except a DPC have been already duplicated.

In this paper, a totally self-checking circuit based on dual-rail MVCM logic is proposed to realize highly reliable VLSI systems with keeping high-speed operation capability. A new non-self-checking (NSC) DPC proposed here has a redundant function because one of the two outputs depends on only a single input in the NSC-DPC. Accordingly, a self-checking (SC) DPC can be designed by the duplication of the proposed NSC-DPC. That is, the dual-rail inputs in one NSC-DPC are shared by the dual-rail inputs in the other one, and where only a single output from each NSC-DPC is used as the resulting dual-rail outputs in the SC-DPC. Consequently, the use of the proposed DPC makes it possible to design a totally self-checking circuit with keeping high driving capability based on dual-rail MVCM logic, which results in small hardware overhead compared with the corresponding NSC circuit based on dual-rail MVCM logic.

To confirm the usefulness of the proposed self-checking circuit, its performance is compared with that of the corresponding full duplication of a dual-rail MVCM circuit in terms of transistor counts, switching delay and dynamic power dissipation under a 0.5-μm standard CMOS technology. As a result, the number of transistors and dynamic power dissipation in the proposed self-checking circuit is reduced to about 67 percent and 68

0-7695-0161-3/99 $10.00 © 1999 IEEE

percent, respectively, in comparison with those of the full duplication of the dual-rail MVCM circuit under the same switching delay.

2. Model of a Multiple-Valued Self-Checking VLSI System

In this section, we describe an MVCM circuit model discussed here, and define a totally self-checking circuit against a single stuck-at fault in the VLSI system

2.1 Basic components of a multiple-valued VLSI

Figure 1 shows three basic components of the proposed MVCM circuit. By the combination of these components, we can realize any multiple-valued combinational circuits. A comparator is to compare an R-valued input X with a threshold value T, and to generate a binary output G where $X, T \in \mathbf{R} = \{0, 1, \cdots, R-1\}$ and $G \in \{0, 1\}$. If X is less than T, G becomes 0. Otherwise, G becomes 1. A DPC is to generate one of three multiple-valued differential-pair outputs (0, 0), (K, 0) and (0, K) in accordance with binary differential-pair inputs where its function is defined in Figure 1. An adder forms arithmetic sum of two multiple-valued inputs A and B where $A, B \in \mathbf{R}$. For example, Figure 2 shows a block diagram of an MVCM circuit whose function is given as

$$Z = \begin{cases} 1 & (X > T), \\ 0 & (X < T). \end{cases} \quad (1)$$

2.2 Definition of a totally self-checking circuit

Self-checking can be defined as the ability to verify automatically, whether there is any fault in logic without the need for externally applied test stimuli. We will consider only malfunctions in the circuit caused by single faults. The concept of a totally self-checking circuit, which can generate a detectable erroneous output for every fault from prescribed set during normal operations, is defined in the following [4]:

Fault Secure (FS): A circuit is fault secure if, for every fault from a prescribed set, the circuit never produces an incorrect code output for code inputs.

Self Testing (ST): A circuit is self-testing if, for every fault from a prescribed set, the circuit produces a non-code output for at least one code input.

Totally Self-Checking (TSC): A circuit is totally self-checking if it is both self-checking and fault secure.

Generally, a duplicated circuit is TSC. Figure 3 shows a self-checking circuit which has a fully duplicated structure of a single-rail MVCM circuit shown in Figure 2. Therefore, its hardware becomes twice as many as that

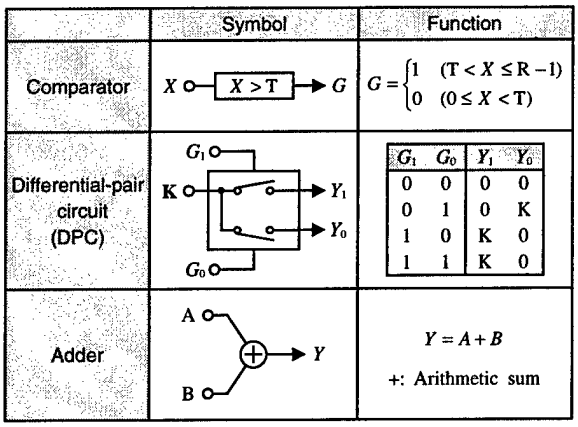


Figure 1. Definition of basic components in an MVCM circuit.

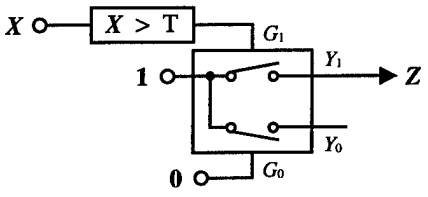


Figure 2. Single-rail MVCM threshold detector.

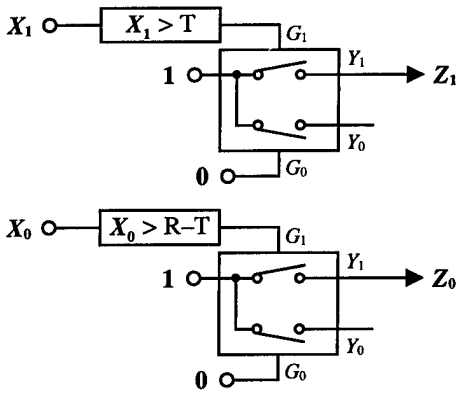


Figure 3. Self-checking circuit with a fully duplicated structure of two single-rail MVCM threshold detectors.

of the circuit shown in Figure 2. In this self-checking R-valued logic circuit, the code words and non-code words are shown in Table 1. The dual-rail code words satisfy the

Table 1. 5-valued dual-rail code.

single-rail	dual-rail	Logic Value	
X	(X_1, X_0)		
0	(0, 4)	"0"	Code word
1	(1, 3)	"1"	
2	(2, 2)	"2"	
3	(3, 1)	"3"	
4	(4, 0)	"4"	
–	otherwise	Non-code word	

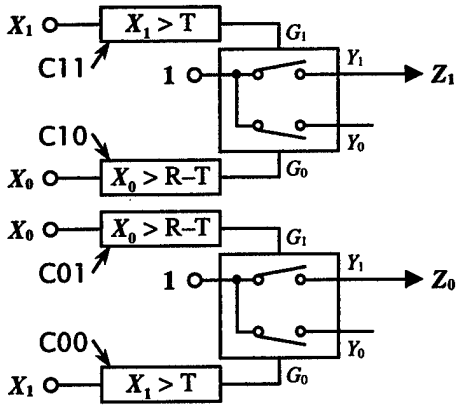


Figure 4. Duplicated dual-rail MVCM threshold detector with complementary logic.

following relation.

$$X_1 + X_0 = R - 1 \tag{2}$$

2.3 High-performance self-checking circuit based on dual-rail MVCM logic

In order to improve the performance of the single-rail MVCM circuit shown in Figure 3, DPCs must be controlled by dual-rail complementary inputs. Figure 4 shows a block diagram of a self-checking circuit using a dual-rail MVCM circuit. Its function is the same as that described by Eq. (1). In this circuit, its switching speed becomes higher than that of the self-checking circuit based on single-rail MVCM logic as shown in Figure 3 because of the large driving capability of DPC. But it requires twice as many input nodes and comparators as the corresponding single-rail MVCM circuit, which causes serious hardware overhead.

Figure 5 shows a block diagram of the proposed self-checking circuit based on dual-rail MVCM logic. Since the functions of two comparators, *C00* and *C01*, are same as two comparators, *C11* and *C10*, respectively, they can be shared to *C11* and *C10* as shown in Figure 5.

Now, let us discuss about the self-checking ability of Figure 5. The DPC has a redundant function, because it depends on only a single input G_1 as shown in Figure 1. As an example, Figure 6 shows a block diagram of the proposed self-checking circuit with a stuck-at-1 fault at the output G_1 of the comparator *C11*. The output of the comparator *C11* is connected to G_{11} and G_{00}. However, the output Z_0 in the *DPC2* depends on only G_{01}, so that *DPC2* operated correctly even when the S-a-1 fault happens at G_{00}. In this way, even if one comparator fails in the circuit of Figure 5, it satisfies the self-checking ability.

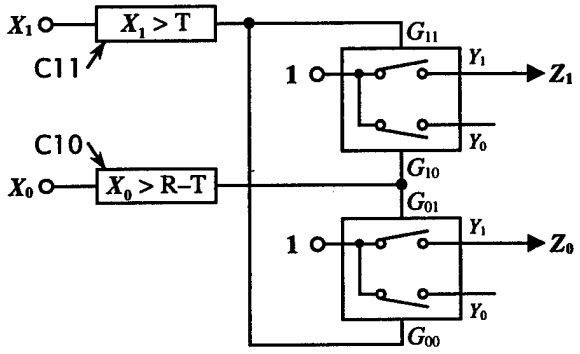


Figure 5. Proposed dual-rail MVCM threshold detector.

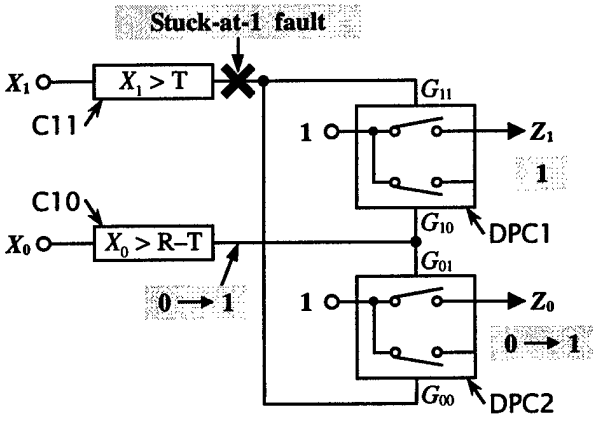


Figure 6. Proposed dual-rail MVCM threshold detector with a stuck-at-1 fault.

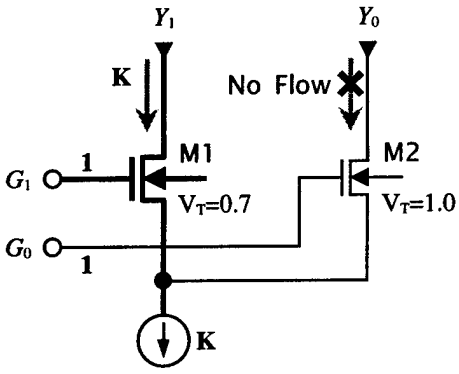


Figure 7. Behavior of the proposed DPC.

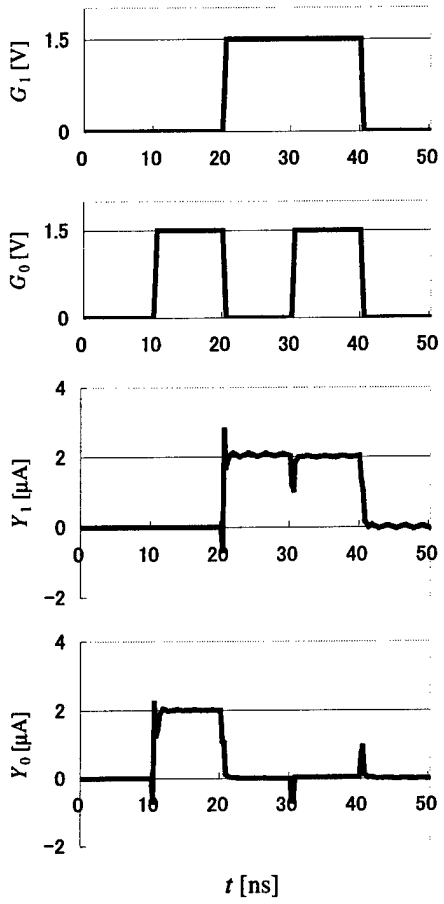


Figure 8. Simulated waveforms of the circuit of Figure 7.

3. Design of a Multiple-Valued Self-Checking Circuit

Figure 7 shows the proposed DPC of Figure 1. The threshold voltage of the nMOS transistor *M1* becomes lower than that of the nMOS transistor *M2*. The lower threshold voltage increases current through transistor. Using this characteristic realizes the selective switching of *M1* when both gate voltages of *M1* and *M2* are same each other by just three transistors.

Figure 8 shows the HSPICE simulation of a DPC. The simulation clearly demonstrates that the expected correct code outputs (K, 0) are appeared when their inputs are (1,1). Figure 9 shows the relationship between the basic components and their circuit diagrams. Current mirror creates replicas of an input *X*. Current source creates a constant value *K*. MVCM circuits consist of these five components shown in Figure 9.

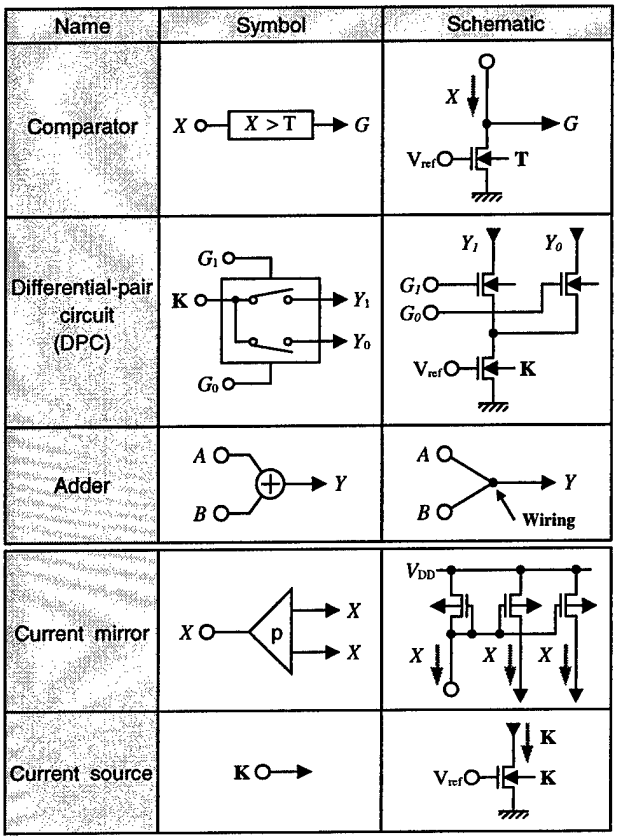
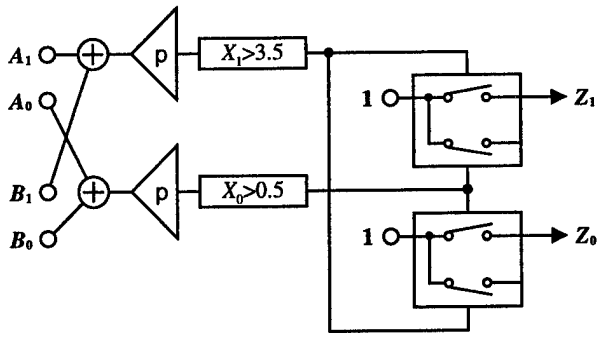


Figure 9. Basic building blocks.

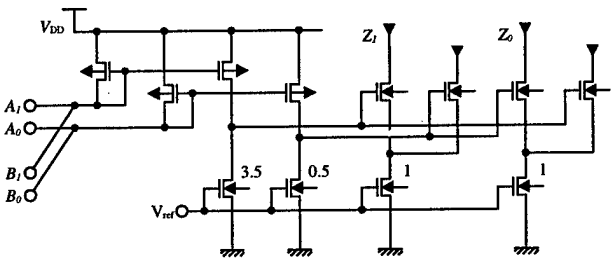
4. Evaluation

Figure 10 shows a circuit diagram of the proposed self-checking circuit shown in Figure 5. The output of the comparator is distributed to two DPCs, so that a compact self-checking circuit can be designed because of no additional comparators for dual-rail MVCM logic.

Table 2 summarizes the comparison of performances among three kinds of MVCM circuits using a



(a) Block diagram



(b) Schematic

Figure 10. Self-checking MVCM threshold detector.

conventional non-self-checking design, a fully duplicated design, and the proposed design, respectively. The use of the proposed NSC-DPC makes it possible to reduce the number of comparators in a self-checking circuit. As a result, in case of the same switching delay, the power dissipation and the number of transistors of the self-checking circuit using the proposed NSC-DPCs are reduced to about 68 percent and 67 percent, respectively, in comparison with those of the corresponding full duplication of an NSC dual-rail MVCM circuit under a 0.5-μm standard CMOS technology.

5. Conclusion

A new compact NSC-DPC based on dual-rail MVCM logic has been proposed, and is useful for the realization of a totally self-checking system with keeping high speed operations at a lower supply voltage. In fact, its performance is superior to that of the equivalent full duplication of an NSC dual-rail MVCM circuit in terms of transistor counts and power dissipation under the same switching delay.

Although the use of the transistor with a higher threshold voltage makes it possible to design a compact DPC with a redundant function, it increases the switching

Table 2. Comparison of three MVCM circuits.

	Non-self-checking MVCM circuit	Fully duplicated MVCM circuit	Proposed MVCM circuit
TSC	No	Yes	Yes
Supply Voltage [V]	1.5	1.5	1.5
Delay [ns]	3.5	3.5	3.5
Power dissipation [μW]	8.3	16.7	11.4
Number of Transistors	9	18	12

delay of the transistor in comparison with that of a normal transistor.

As a future problem, it is also important to design a DPC with small overhead of a switching delay and keeping small transistor counts.

References

[1] R. K. Watts, Ed., "*Submicron Integrated Circuits*," Wiley Interscience, New York, 1989.

[2] S.Malhi and P.Chatterjee: "1-V Microsystems – Scaling on Schedule for Personal Communications," *IEEE Circuits and Devices*, 10, 2, pp.13-17, 1994.

[3] Foty and E.J.Nowak: "MOSFET Technology for Low-Voltage/Low-Power Applications," *IEEE Micro*, 14, 3, pp.68-76, 1994.

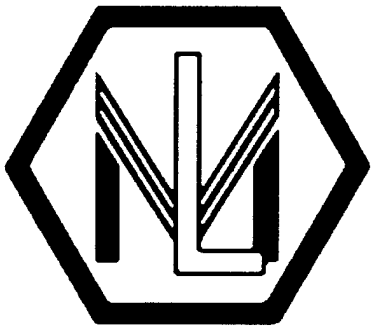
[4] D. A. Anderson and G. Metze, "Design of Totally Self-Checking Check Circuits for m-Out-of-n Codes," *IEEE Trans. Comput.*, Vol.C-22, No.3, pp.263-269, Mar. 1973.

[5] J. F. Wakerly, "*Error Detecting Codes: Self-Checking Circuits and Applications*," Elsevier North-Holland, New York, 1978.

[6] T. Hanyu and M. Kameyama, "A 200 MHz Pipelined Multiplier Using 1.5V-Supply Multiple-Valued MOS Current-Mode Circuits with Dual-Rail Source-Coupled Logic," *IEEE J. Solid-State Circuits*, Vol.30, No.11, pp.1239-1245, Nov. 1995.

[7] T. Hanyu, S. Kazama and M. Kameyama, "Design and Implementation of a Low-Power Multiple-Valued Current-Mode Integrated Circuit with Current-Source Control," *IEICE Trans. Electron.*, Vol. E80-C, No.7, pp.941-947, July 1997.

SESSION IXB
FUZZY LOGIC
CHAIR: Okihiko Ishizuka



On the Concept of Qualitative Fuzzy Set*

Helmut Thiele
University of Dortmund, Department of Computer Science I,
D-44221 Dortmund, Germany
Phone: +49 231 755 6152
Fax: +49 231 755 6555
E-Mail: thiele@ls1.informatik.uni-dortmund.de

Abstract

Following T. Y. LIN *vague concepts such as "high" or "small" ("amount of money" as linguistic variable) should be described by classes of "equivalent" fuzzy sets on a fixed suitable universe U. Furthermore* LIN *proposed to generate such classes by factorization of the fuzzy power set* $\mathrm{FI\!P}(U)$ *of U with respect to a given equivalence relation* $\approx$ *on* $\mathrm{FI\!P}(U)$. *By three examples we show that this approach is not successful if we intend to shift algebraic operations with fuzzy sets to classes generated by equivalence relations. Using a* KRIPKE*-style semantics approach we can introduce a sensible concept of a qualitative fuzzy set and, furthermore, sensible algebraic operations with such sets. The concept of multi-fuzzy set introduced by* LIN *is a special case of our concept of context-depending fuzzy set.*

Keywords: Fuzzy sets, vague concepts, qualitative fuzzy sets, multi-fuzzy sets.

1. Introduction

We start with recalling some mathematical notions and notations.

By $\mathrm{I\!R}$ and $[0, 1]$ we denote the set of all real numbers and the set of all real numbers r with $0 \le r \le 1$, respectively.

Let U be a fixed non-empty set called universe. The power set of U is denoted by $\mathrm{I\!P}(U)$. Furthermore, we will use the usual denotations of crisp set theory, for instance, $A \cap B$, $A \cup B$, and $A \times B$ for the intersection, union, and the CARTESIan product of the crisp sets A and B, respectively. Especially, by A^n we denote the n-fold CARTESIan product of A ($n \ge 1$, n integer).

In the following we shall use the well-known λ-operator. Assume A, B, and C are crisp sets and $f : A \times B \to C$. Then $\lambda a f(a, b)$ denotes the mapping $\varphi : A \to C$ that is defined by

$$\varphi(a) =_{def} f(a, b)$$

where $b \in B$ is fixed and a runs through A. The mapping $\lambda b f(a, b)$ is defined analogously.

A fuzzy set F on U is a mapping $F : U \to [0, 1]$, i. e. we do not distinguish between a fuzzy set F on U and its membership function μ_F. The set $\{F | F : U \to [0, 1]\}$ is called fuzzy power set of U and denoted by $\mathrm{FI\!P}(U)$. For $C \subseteq [0, 1]$ we define $F^{-1}(C) =_{def} \{x | x \in U \wedge F(x) \in C\}$.

An n-ary operation ($n \ge 1$, n integer) on $[0, 1]$ is a mapping $op : [0, 1]^n \to [0, 1]$ where $[0, 1]^n$ denotes the n-fold CARTESIan product of $[0, 1]$.

As usual we "shift" a given operation op to an operation Op for fuzzy sets $F_1, \ldots, F_n$ on U as follows where $x \in U$:

$$Op(F_1, \ldots, F_n)(x) =_{def} op(F_1(x), \ldots, F_n(x)).$$

In daily life we can come across a lot of "computing rules" having the form

"high plus small is high"

or

"many minus small is many".

Such rules can be denoted as rules for computing with "vague quantities" or with "fuzzy qualities" or, interpreting proposals made by L. A. ZADEH [30, 32, 34, 35], as rules for "computing with words". In the papers [24, 25] we have called such rules "computing with words in the *narrow* sense" while by "computing with words in the *wide* sense" we understand the use of vague concepts in (approximate) reasoning, in modelling, and in programming.

In the framework of the usual fuzzy arithmetics a rule of the form "high plus small is high" should be interpreted and evaluated as follows:

1. Take as the universe U the set $\mathrm{I\!R}$ of all real numbers
2. interpret the words "high" and "small" by real fuzzy numbers θ and σ, respectively

*This research was supported by the Deutsche Forschungsgemeinschaft as part of the Collaborative Research Center "Computational Intelligence" (SFB 531)

0-7695-0161-3/99 $10.00 © 1999 IEEE

3. interpret the word "plus" by an addition + for real fuzzy numbers, for instance, if α and β are arbitrary fuzzy numbers then + is defined by

   $(\alpha+\beta)(x) =_{def} \sup\{\alpha(y)+\beta(z) | y,z \in \mathbb{R} \wedge x = y+z\}$

   for every $x \in \mathbb{R}$

4. interpret the word "is" by an identity relation = for real fuzzy numbers

5. make sure that the equation $\theta + \sigma = \theta$ holds.

Now, we have to state that the considerations above highly depend on the context. Take, for instance, the linguistic variable "amount of money" and for this variable the linguistic terms "high" and "small". Then for an oil sheik or "Deutsche Bank" the words "high" and "small" are to be interpreted, for instance, by real fuzzy numbers describing an amount of about 1 billion dollars and 500 thousand dollars ("peanuts"), respectively, while for an unemployed person these words are to be interpreted, for instance, by real fuzzy numbers describing an amount of about 500 dollars and 2 dollars, respectively.

From the discussion above we derive the following statements and claims: Assume we have fixed U as $\mathbb{R}$ and the domain of all real fuzzy numbers as possible interpretations of the words "high" and "small". Furthermore assume that we have fixed an interpretation of the words "plus" and "is". Then

1. Consider all interpretations of the words "high" and "small" such that the equation

   "high plus small is high"

   holds.

2. Consider all such interpretations of the word "high" as a class of "equivalent" (A. KANDEL uses the word "admissible function" [7] while T. Y. LIN prefers to speak about "equivalent" fuzzy sets [12]) real fuzzy numbers and call this class a "qualitative" real fuzzy number and consider this class as a "qualitative" interpretation of the word "high". Do the same for the word "small".

3. We intend to develop a "qualitative" interpretation, illustrated by the equation

   "high plus small is high"

   as follows:

   3.1. Interpret the words "high" and "small" by classes of "equivalent" real fuzzy numbers.

   3.2. Define an addition and an identity for classes such that the equation above holds.

On the basis of our knowledge from classical algebra one could think that

- the "equivalence classes" could be generated by equivalence relations for fuzzy sets
- the operations could be defined as for factor structures in classical algebra.

In the next chapter we show by three examples that this approach is not successful, in general.

2. Three counterexamples for the classical "factorization" approach

Counterexample 1

For definiteness we recall some notions and notations of the general topology, but in a simplified form and adapted to the given case.

By $CLOS([0,1])$ we denote the set of all sets $C \subseteq [0,1]$ that are closed with respect to the usual topology in $[0,1]$.

Assume $\mathfrak{T} \subseteq \mathbb{P}(U)$.

Definition 2.1

$\mathfrak{T}$ is said to be a topology on U

$=_{def}$ 1. $\forall \mathfrak{S}(\mathfrak{S} \subseteq \mathfrak{T} \rightarrow \bigcap \mathfrak{S} \in \mathfrak{T})$

2. $\forall S \forall T(S, T \in \mathfrak{T} \rightarrow S \cup T \in \mathfrak{T})$

Assume $F : U \rightarrow [0,1]$. Then one defines the topology $TOP(F,U)$ that is generated by F on U as follows.

Definition 2.2

$$TOP(F,U) =_{def} \bigcap \left\{ \mathfrak{T} \middle| \begin{array}{l} \{F^{-1}(C) | C \in CLOS([0,1])\} \subseteq \mathfrak{T} \\ \wedge \mathfrak{T} \text{ is a topology on } U \end{array} \right\}$$

Obviously, $TOP(F,U)$ is the "smallest" topology $\mathfrak{T}$ on U such that the function F is continuous with respect to $\mathfrak{T}$.

Assume $h : [0,1] \rightarrow [0,1]$.

Definition 2.3 (see LIN [12])

h is said to be a relative self-homeomorphism of $[0,1]$

$=_{def}$ 1. *h is a bijection between $[0,1]$ and $[0,1]$*

2. *h and h^{-1} are continuous*

3. *$h(0) = 0$ and $h(1) = 1$.*

Assume $F, G : U \rightarrow [0,1]$.

Definition 2.4 (see LIN [12])

$F \underset{L}{\approx} G$

$=_{def}$ 1. *$TOP(F,U) = TOP(G,U)$ and*

2. *there exists a self-homeomorphism h such that for every $x \in U$*

$$F(x) = h(G(x)).$$

Obviously, $\underset{L}{\approx}$ is an equivalence relation on $F\mathbb{P}(U)$. Hence we can construct the set $F\mathbb{P}(U)/\underset{L}{\approx}$ of "LIN-equivalence classes" with respect to $\underset{L}{\approx}$. We call classes $\mathfrak{C} \in F\mathbb{P}(U)/\underset{L}{\approx}$ "LIN-qualitative fuzzy sets".

Now, we shall show that very important operations with fuzzy sets $F, G \in \mathrm{FI\!P}(U)$ cannot be shifted to operations with LIN-qualitative fuzzy sets.

Consider the ŁUKASIEWICZ disjunction $vel_{Ł}$ defined by

$$vel_{Ł}(r,s) = \min(1, r+s)$$

for $r, s \in [0, 1]$. Furthermore, consider the operation $Vel_{Ł}$ for $F, G \in \mathrm{FI\!P}(U)$ defined by

$$Vel_{Ł}(F,G)(x) =_{def} \min(1, F(x)+G(x))$$

for $x \in U$.

Now we construct a universe U and fuzzy sets F, G, and H on U such that

1. $F \underset{L}{\approx} G$ but
2. not $Vel_{Ł}(F,H) \underset{L}{\approx} Vel_{Ł}(G,H)$.

This result means that the operation $Vel_{Ł}$ which is very important for studying MV-algebras cannot be shifted from fuzzy sets on U to LIN-qualitative fuzzy sets on U, i. e. classes from $\mathrm{FI\!P}(U)/\underset{L}{\approx}$.

To prove this result we put $U =_{def} \{a, b, c\}$ with $a \neq b \wedge b \neq c \wedge c \neq a$,

$$\begin{array}{lll} F(a) =_{def} 0 & G(a) =_{def} 0 & H(a) =_{def} 0 \\ F(b) =_{def} \frac{3}{4} & G(b) =_{def} \frac{1}{4} & H(b) =_{def} \frac{1}{4} \\ F(c) =_{def} 1 & G(c) =_{def} 1 & H(c) =_{def} 1 \end{array}$$

and

$$h(r) =_{def} \begin{cases} 3 \cdot r & \text{if } 0 \leq r \leq \frac{1}{4} \\ \frac{r}{3} + \frac{2}{3} & \text{if } \frac{1}{4} < r \leq 1 \end{cases}$$

Then by definition we get

$$\mathrm{I\!P}(U) = TOP(F,U) = TOP(G,U).$$

Furthermore, we have that the function h defined above is a relative self-homeomorphism of $[0, 1]$ and

$$\forall u(u \in U \rightarrow F(u) = h(G(u))).$$

Hence, we have $F \underset{L}{\approx} G$.

Now, we get

$$Vel_{Ł}(F,H)(b) = \min(1, F(b)+H(b)) = \min\left(1, \tfrac{3}{4}+\tfrac{1}{4}\right) = 1$$

$$Vel_{Ł}(G,H)(b) = \min(1, G(b)+H(b)) = \min\left(1, \tfrac{1}{4}+\tfrac{1}{4}\right) = \tfrac{1}{2}.$$

Now, we assume

$$Vel_{Ł}(F,H) \underset{L}{\approx} Vel_{Ł}(G,H),$$

so we get

$$Vel_{Ł}(G,H) \underset{L}{\approx} Vel_{Ł}(F,H),$$

hence there exists a relative self-homeomorphism h' of U such that

$$\forall u(u \in U \rightarrow Vel_{Ł}(G,H)(u) = h'(Vel_{Ł}(F,H)(u))).$$

For $u = b$ we have

$$Vel_{Ł}(G,H)(b) = \tfrac{1}{2}$$

$$h'(Vel_{Ł}(F,H)(b)) = h'(1) = 1,$$

hence we get the contradiction

$$\tfrac{1}{2} = 1.$$

Counterexample 2

One could think that counterexample 1 works because we have chosen the function $vel_{Ł}$. In the following we are going to show that the same holds if we replace $vel_{Ł}$ by the essentially simpler function min. We put

$$U =_{def} \{a, b, c, d\} \text{ where } a \neq b,\ b \neq c,\ c \neq d, \text{ and } d \neq a$$

$$\begin{array}{lll} F(a) =_{def} 0 & G(a) =_{def} 0 & H(a) =_{def} 0 \\ F(b) =_{def} \frac{7}{10} & G(b) =_{def} \frac{5}{10} & H(b) =_{def} \frac{2}{10} \\ F(c) =_{def} \frac{1}{10} & G(c) =_{def} \frac{4}{10} & H(c) =_{def} \frac{2}{10} \\ F(d) =_{def} 1 & G(d) =_{def} 1 & H(d) =_{def} 1 \end{array}$$

Then we get

$$TOP(F) = TOP(G) = \mathrm{I\!P}(U).$$

Furthermore, it is trivial that there exists a self-homeomorphism h of $[0, 1]$ such that for every $x \in U$ the equation $F(x) = h(G(x))$ holds, hence $F \underset{L}{\approx} G$.

Now, we obtain

$$\begin{array}{ll} \min(F(a), H(a)) = 0 & \min(G(a), H(a)) = 0 \\ \min(F(b), H(b)) = \frac{2}{10} & \min(G(b), H(b)) = \frac{2}{10} \\ \min(F(c), H(c)) = \frac{1}{10} & \min(G(c), H(c)) = \frac{2}{10} \\ \min(F(d), H(d)) = 1 & \min(G(d), H(d)) = 1 \end{array}$$

Obviously, we get $Min(F,H) \not\underset{L}{\approx} Min(G,H)$ because there is no self-homeomorphism h such that for every $x \in U$ the equation $Min(F,H)(x) = h(Min(G,H)(x))$ holds.

Counterexample 3

Now, we discuss an example which is more lucid than the examples 1 and 2 coming from LIN's concept of a qualitative fuzzy set.

Our example will very instructively demonstrate that the concept of qualitative fuzzy set based on an equivalence relation $\approx$ on $\mathrm{FI\!P}(U)$ is not suitable for constructing qualitative fuzzy sets by "factorization" with respect to $\approx$ in the sense of classical algebra.

Assume that the universe used is the set $\mathrm{I\!R}$ of all real numbers, i. e. we put

$$U =_{def} \mathrm{I\!R}.$$

Assume F and G are fuzzy sets on U, i. e. $F, G : \mathrm{I\!R} \rightarrow [0, 1]$.

Definition 2.5

$F \approx G =_{def} \exists c(c \in \mathbb{R} \wedge \forall x(x \in \mathbb{R} \rightarrow F(x) = G(x+c)))$

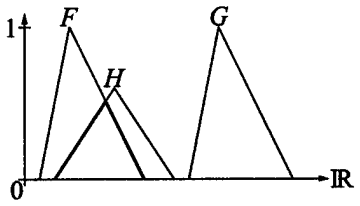
This definition means that F and G are equivalent if and only if they are congruent modulo a translation along the real axis.

Obviously, $\approx$ is an equivalence relation on $\mathrm{F}\mathbb{P}(\mathbb{R})$.

Now, for arbitrary fuzzy sets F and H on $\mathbb{R}$ define the operation $Min(F,H)$ as follows where $x \in \mathbb{R}$

$$Min(F,H)(x) =_{def} \min(F(x), H(x)).$$

Then the following picture shows that $F \approx G$ does not imply $Min(F,H) \approx Min(G,H)$, in general.



Obviously, we have $F \approx G$, but $Min(F,H) \neq \emptyset$ and $Min(G,H) = \emptyset$, hence $Min(F,H) \not\approx Min(G,H)$.

3. A KRIPKE-style semantics approach to a sensible concept of qualitative fuzzy sets

Let W be a non-empty set of possible worlds.

Definition 3.1

Φ is said to be a context-depending fuzzy set on U with respect to W

$=_{def} \Phi : W \times U \rightarrow [0,1]$

At the first glance a context-depending fuzzy set on U with respect to W can be considered as a usual fuzzy set on the CARTESIan product $W \times U$ or as a binary fuzzy relation between the sets W and U.

The key point of our following approach consists of using a function $\Phi : W \times U \rightarrow [0,1]$ for describing a "vague concept" on U, for instance, the vague concept "high" (amount of money).

If we put $W =_{def} \{1, \ldots, n\}$ ($n \geq 1$, n integer) then our concept of a context-depending fuzzy set $\Phi : W \times U \rightarrow [0,1]$ (on U with respect to W) coincides with LIN's concept of multi-fuzzy set (see [14]).

Strictly speaking, a given context-depending fuzzy set $\Phi : W \times U \rightarrow [0,1]$ generates a class $\mathfrak{C}_\Phi$ of "equivalent" fuzzy sets (or "admissible" fuzzy sets) on U as follows.

Definition 3.2

$\mathfrak{C}_\Phi =_{def} \{\lambda u \Phi(w,u) | w \in W\}$

Now, we consider the fuzzy power set $\mathrm{F}\mathbb{P}(W \times U)$. Using the concepts introduced by definition 3.1 and 3.2 we introduce a new "factorization" of $\mathrm{F}\mathbb{P}(U)/_W$ of the "classical" power set $\mathrm{F}\mathbb{P}(U)$ as follows.

Definition 3.3

$\mathrm{F}\mathbb{P}(U)/_W =_{def} \{\mathfrak{C}_\Phi | \Phi : W \times U \rightarrow [0,1]\}$

Comparing the construction $\mathrm{F}\mathbb{P}(U)/_W$ with the classical construction $\mathrm{F}\mathbb{P}(U)/_\approx$ where $\approx$ is an equivalence relation on $\mathrm{F}\mathbb{P}(U)$ we can state:

While $\mathrm{F}\mathbb{P}(U)/_\approx$ is a partition of $\mathrm{F}\mathbb{P}(U)$ we are faced with the following two important facts:

1. $\mathrm{F}\mathbb{P}(U)/_W$ is not a covering, in general
2. $\mathrm{F}\mathbb{P}(U)/_W$ is not disjoint, in general.

The fact that $\mathrm{F}\mathbb{P}(U)/_W$ is not a covering must be interpreted so that there exists a fuzzy set $F \in \mathrm{F}\mathbb{P}(U)$ which is not an "admissible" fuzzy set for any class $\mathfrak{C} \in \mathrm{F}\mathbb{P}(U)/_W$, i. e. which is not an "admissible" fuzzy set for any "vague concept" $\Phi : W \times U \rightarrow [0,1]$, i. e. for an arbitrary "vague concept" $\Phi : W \times U \rightarrow [0,1]$ there is no world $w \in W$ such that $F = \lambda u \Phi(w,u)$.

The fact that $\mathrm{F}\mathbb{P}(U)/_W$ can be not disjoint means that there can exist "vague concepts" $\Phi : W \times U \rightarrow [0,1]$ and $\Psi : W \times U \rightarrow [0,1]$ such that

1. $\mathfrak{C}_\Phi \neq \mathfrak{C}_\Psi$ and
2. $\mathfrak{C}_\Phi \cap \mathfrak{C}_\Psi \neq \emptyset$.

From conclusion 1 we get that

3. $\exists w(w \in W \wedge \forall w'(w' \in W \rightarrow \lambda u \Phi(w,u) \neq \lambda u \Psi(w',u)))$ or $\exists w'(w' \in W \wedge \forall w(w \in W \rightarrow \lambda u \Psi(w',u) \neq \lambda u \Phi(w,u)))$.

From conclusion 2 we get that

4. $\exists w \exists w'(w, w' \in W \wedge \lambda u \Phi(w,u) = \lambda u \Psi(w',u))$.

If we interpret the vague concept "high" (amount of money) by Φ and the vague concept "small" (amount of money) by Ψ then the condition $\mathfrak{C}_\Phi \neq \mathfrak{C}_\Psi$ means that these vague concepts are different.

Furthermore, the conclusion 4 could be interpreted that the same fuzzy set $F \in \mathrm{F}\mathbb{P}(U)$ for an unemployed person w describes a high amount, i. e. $F = \lambda u \Phi(w,u)$, while for an oil sheik w' the same fuzzy sets describes a small amount (of money), i. e. $F = \lambda u \Psi(w',u)$, for instance.

Now, we shall discuss how operations

$$Op^n : \mathrm{F}\mathbb{P}(U)^n \rightarrow \mathrm{F}\mathbb{P}(U) \qquad (n \geq 1, n \text{ integer})$$

can be "lifted" to operations

$$OP^n : (\mathrm{F}\mathbb{P}(U)/_W)^n \rightarrow \mathrm{F}\mathbb{P}(U)/_W.$$

Using the terminology and "philosophy" of classical algebra in constructing factor structures from given algebraic structures (groups, rings, lattices, ...) for $\Phi, \Psi : W \times U \rightarrow [0,1]$ and $\mathfrak{C} \in \mathrm{F}\mathbb{P}(U)/_W$ we define:

Definition 3.4

1. *Φ is said to be a representative of the class $\mathfrak{C}$* $=_{def} \mathfrak{C} = \mathfrak{C}_\Phi$
2. *Φ and Ψ are said to be equivalent (shortly: $\Phi \approx \Psi$* $=_{def} \mathfrak{C}_\Phi = \mathfrak{C}_\Psi$

Proposition 3.1

The relation $\approx$ is an equivalence relation on $\mathrm{F}\mathbb{P}(W \times U)$.

Now we are going to discuss how operation

$$OP^n : \mathrm{F}\mathbb{P}(U)^n \to \mathrm{F}\mathbb{P}(U)$$

can be lifted to operations

$$OP^n : (\mathrm{F}\mathbb{P}(U)/_W)^n \to \mathrm{F}\mathbb{P}(U)/_W.$$

Assume $\mathfrak{C}_1, \ldots, \mathfrak{C}_n \in \mathrm{F}\mathbb{P}(U)/_W$ and $\Phi_1, \ldots, \Phi_n : W \times U \to [0,1]$ are representatives of these classes, i. e. we have

$$\mathfrak{C}_1 = \mathfrak{C}_{\Phi_1}, \ldots, \mathfrak{C}_n = \mathfrak{C}_{\Phi_n}.$$

Now, using the "philosophy" of classical algebra we try to define the result

$$OP^n(\mathfrak{C}_1, \ldots, \mathfrak{C}_n)$$

of the operation OP^n applied to the arguments $\mathfrak{C}_1, \ldots, \mathfrak{C}_n$ by using the representatives $\Phi_1, \ldots, \Phi_n$ and the operation Op^n. From the classical algebra we know the important claim that we have to prove the independence of $OP^n(\mathfrak{C}_1, \ldots, \mathfrak{C}_n)$ from the choice of the representatives $\Phi_1, \ldots, \Phi_n$ of the classes $\mathfrak{C}_1, \ldots, \mathfrak{C}_n$.

It is surprising and for the following very important that this assertion can be proved for 1-ary operations

$$Op : \mathrm{F}\mathbb{P}(U) \to \mathrm{F}\mathbb{P}(U)$$

without any difficulties and additional assumptions.

In contrast to this fact in the n-ary case for $n \geq 2$ (n integer) we are faced with serious difficulties that can be overcome only by additional assumptions, in general.

We underline that this situation as a matter of principle differs from the situation which we know from the "factorization theory" in classical algebra.

The case of 1-ary operations

Assume

$$Op : \mathrm{F}\mathbb{P}(U) \to \mathrm{F}\mathbb{P}(U),$$
$$\mathfrak{C} \in \mathrm{F}\mathbb{P}(U)/_W,$$
$$\Phi, \Psi : W \times U \to [0,1]$$

with $\mathfrak{C} = \mathfrak{C}_\Phi$.

Definition 3.5

$OP_\Phi(\mathfrak{C}) =_{def} \{Op(\lambda u \Phi(w,u)) | w \in W\}$

Theorem 3.2

1. $OP_\Phi(\mathfrak{C}) \in \mathrm{F}\mathbb{P}(U)/_W$
2. *If* $\mathfrak{C}_\Phi = \mathfrak{C}_\Psi$ *then* $OP_\Phi(\mathfrak{C}) = OP_\Psi(\mathfrak{C})$.

Remarks

1. Theorem 3.2 means that $OP_\Phi(\mathfrak{C})$ is independent of Φ if $\mathfrak{C} = \mathfrak{C}_\Phi$.
2. $OP_\Phi(\mathfrak{C})$ can also be considered as a mapping from $\mathrm{F}\mathbb{P}(W \times U)/_\approx$ into $\mathrm{F}\mathbb{P}(W \times U)/_\approx$.

The case of n-ary Operations $Op : \mathrm{F}\mathbb{P}(U)^n \to \mathrm{F}\mathbb{P}(U)$ where $n \geq 2$, n integer

Without loss of generality we can assume that $n = 2$. Furthermore, we assume

$$Op : \mathrm{F}\mathbb{P}(U) \times \mathrm{F}\mathbb{P}(U) \to \mathrm{F}\mathbb{P}(U),$$
$$\mathfrak{C}_1, \mathfrak{C}_2 \in \mathrm{F}\mathbb{P}(U)/_W,$$
$$\Phi_1, \Phi_2, \Psi_1, \Psi_2 : W \times U \to [0,1],$$
$$\mathfrak{C}_1 = \mathfrak{C}_{\Phi_1}, \mathfrak{C}_2 = \mathfrak{C}_{\Phi_2}$$

Definition 3.6

$OP_{\Phi_1,\Phi_2}(\mathfrak{C}_1, \mathfrak{C}_2) =_{def} \{Op(\lambda u \Phi_1(w,u), \lambda u \Phi_2(w,u)) | w \in W\}$

Theorem 3.3

1. $OP_{\Phi_1,\Phi_2}(\mathfrak{C}_1, \mathfrak{C}_2) \in \mathrm{F}\mathbb{P}(U)/_W$
2. *The implication*

   If $\mathfrak{C}_{\Phi_1} = \mathfrak{C}_{\Psi_1}$ *and* $\mathfrak{C}_{\Phi_2} = \mathfrak{C}_{\Psi_2}$ *then* $OP_{\Phi_1,\Phi_2}(\mathfrak{C}_1, \mathfrak{C}_2) = OP_{\Psi_1,\Psi_2}(\mathfrak{C}_1, \mathfrak{C}_2)$

   does not hold, in general.

Now, we are going to introduce an additional assumption such that theorem 3.3 holds.

From the assumptions

$$\mathfrak{C}_{\Phi_1} = \mathfrak{C}_{\Psi_1}$$

and

$$\mathfrak{C}_{\Phi_2} = \mathfrak{C}_{\Psi_2}$$

we obtain (see the proof of theorem 3.2) that there exist mappings $\alpha_1, \beta_1, \alpha_2, \beta_2 : W \to W$ such that for $i \in \{1,2\}$,

$$\forall w(w \in W \to \lambda u \Phi_i(w,u) = \lambda u \Psi_i(\alpha_i(w), u))$$

and

$$\forall w'(w' \in W \to \lambda u \Psi_i(w',u) = \lambda u \Phi_i(\beta_i(w'), u))$$

hold.

Now, we assume that the mappings described above can be chosen such that $\alpha_1 = \alpha_2$ and $\beta_1 = \beta_2$, i. e. we define

Definition 3.7

$[\Phi_1, \Psi_1]$ *and* $[\Phi_2, \Psi_2]$ *have a common* SKOLEM*ization* $=_{def}$ *There are mappings* $\alpha, \beta : W \to W$ *such that for* $i \in \{1,2\}$,

$$\forall w(w \in W \to \lambda u \Phi_i(w,u) = \lambda u \Psi_i(\alpha(w), u))$$

and

$$\forall w'(w' \in W \to \lambda u \Psi_i(w',u) = \lambda u \Phi_i(\beta(w'), u))$$

hold.

Then we have

Theorem 3.4

If 1. $\mathfrak{C}_1 = \mathfrak{C}_{\Phi_1} = \mathfrak{C}_{\Psi_1}$ *and* $\mathfrak{C}_2 = \mathfrak{C}_{\Phi_2} = \mathfrak{C}_{\Psi_2}$

2. $[\Phi_1, \Psi_1]$ *and* $[\Phi_2, \Psi_2]$ *have a common* SKOLEM*ization then* $OP_{\Phi_1,\Phi_2}(\mathfrak{C}_1, \mathfrak{C}_2) = OP_{\Psi_1,\Psi_2}(\mathfrak{C}_1, \mathfrak{C}_2)$.

Acknowledgement

The author wishes to thank STEPHAN LEHMKE for fruitful scientific discussions and ULRICH FIESELER for his help in preparing the manuscript.

References

[1] P. M. Cohn. *Universal algebra.* Harper and Row (New York, Evanston and London) and John Weatherhill, Inc. (Tokyo), 1965. 333 pages.

[2] D. Dubois and H. Prade. Rough fuzzy sets and fuzzy rough sets. *International Journal of General Systems*, 17:191–209, 1990.

[3] *EUFIT '98—Sixth European Congress on Intelligent Techniques and Soft Computing*, Aachen, Germany, Sept. 7–10, 1998.

[4] *FUZZ-IEEE '96— Fifth IEEE International Conference on Fuzzy Systems*, New Orleans, USA, Sept. 8–11, 1996.

[5] *International Panel Conference on Soft and Intelligent Computing*, Budapest, Hungary, Oct. 7–10, 1996.

[6] *Proceedings of the Second Annual Joint Conference on Information Science*, Wrightsville Beach, North Carolina, Sept. 28–Oct. 1, 1995.

[7] A. Kandel. *Fuzzy Mathematical Techniques with Applications.* Addison Wesley, Reading, Massachusetts, 1986.

[8] H. Kiendl. *Fuzzy Control methodenorientiert.* R. Oldenbourg Verlag, München, Wien, 1997.

[9] B. Kuipers. Qualitative simulation. *Artificial Intelligence*, 29, 1986.

[10] S. Lehmke. Personal communication.

[11] T. Y. Lin. Context free fuzzy sets. In JCIS '95 [6], pages 518–521.

[12] T. Y. Lin. Neighborhood systems — a qualitative theory for fuzzy and rough sets. In JCIS '95 [6], pages 255–258.

[13] T. Y. Lin. A set theory for soft computing—a unified view of fuzzy sets via neighborhoods. In FUZZ-IEEE '96 [4], pages 1140–1146.

[14] T. Y. Lin. Context free fuzzy sets and information tables. In EUFIT '98 [3], pages 76–80.

[15] T. Y. Lin. Granular fuzzy sets. In EUFIT '98 [3], pages 94–98.

[16] M. Mareš. *Computation Over Fuzzy Quantities.* CRC Press, Boca Raton, 1994.

[17] M. Mareš. How much is "many" minus "several"? In IPCSIC '96 [5], pages 187–192.

[18] M. Mareš. Weak arithmetics of fuzzy numbers. *Fuzzy Sets and Systems*, 91:143–153, 1997.

[19] A. Marková-Stupňanová. A note to the addition of fuzzy numbers based on a continuous archimedean T-norm. *Fuzzy Sets and Systems*, 91:253–258, 1997.

[20] V. Novák. Is crucial role in soft computing played by words? In IPCSIC '96 [5], pages 223–228.

[21] W. Pedrycz. Granular computing in fuzzy modeling and data mining. Talk held at the University of Dortmund, Feb. 24, 1998.

[22] H. Thiele. Fuzzy rough sets versus rough fuzzy sets — an interpretation and a comparative study using concepts of modal logic. In *EUFIT '97—Fifth European Congress on Intelligent Techniques and Soft Computing*, volume 1, pages 159–167, Aachen, Germany, Sept. 8–11, 1997.

[23] H. Thiele. Fuzzy rough sets versus rough fuzzy sets — an interpretation and a comparative study using concepts of modal logic. Technical Report CI-30/98, University of Dortmund, Collaborative Research Center 531, Apr. 1998. Extended version of [22].

[24] H. Thiele. On semantic models for investigating 'computing with words'. In *Second International Conference on Knowledge-Based Intelligent Electronic Systems*, Adelaide, Australia, Apr. 21–23, 1998. Keynote address.

[25] H. Thiele. On semantic models for investigating 'computing with words'. Technical Report CI-32/98, University of Dortmund, Collaborative Research Center 531, Apr. 1998. Extended version of [24].

[26] R. R. Yager, S. Ovchinnikov, R. M. Tong, and H. T. Nguyen, editors. *Fuzzy Sets and Applications — Selected Papers by L. A. Zadeh.* John Wiley & Sons, 1987.

[27] L. A. Zadeh. Outline of a new approach to the analysis of complex systems and decision processes. *IEEE Trans. on Systems, Man and Cybernetics*, 3(1):28–44, 1973. Reprinted in [26].

[28] L. A. Zadeh. The concept of a linguistic variable and its application to approximate reasoning — I. *Information Sciences*, 8:199–249, 1975. Reprinted in [26].

[29] L. A. Zadeh. Fuzzy logic and approximate reasoning. *Synthese*, 30:407–428, 1975.

[30] L. A. Zadeh. Fuzzy sets and information granularity. In M. M. Gupta, R. K. Ragade, and R. R. Yager, editors, *Advances in Fuzzy Set Theory and Applications*, pages 3–18. North-Holland, Amsterdam, New York, Oxford, 1979.

[31] L. A. Zadeh. Test score semantics for natural language and meaning representation via PRUF. In B. B. Rieger, editor, *Empirical Semantics: a collection of new approaches in the field*, volume 12 of *Quantitative Linguistics*, pages 282–349. Studienverlag Brockmeyer, Bochum, 1981.

[32] L. A. Zadeh. Fuzzy logic = computing with words. *IEEE Transactions On Fuzzy Systems*, 4(2):103–111, 1996.

[33] L. A. Zadeh. Fuzzy logic and the calculi of fuzzy rules and fuzzy graphs: A precis. *Multiple Valued Logic — An International Journal*, 1(1), 1996.

[34] L. A. Zadeh. The Key Roles of Information Granulation and Fuzzy Logic in Human Reasoning, Concept Formulation and Computing with Words. In FUZZ-IEEE '96 [4].

[35] L. A. Zadeh. Toward a theory of fuzzy information granulation and its centrality in human reasoning and fuzzy logic. *Fuzzy Sets and Systems*, 90:111–127, 1997.

On Some Classes of Fuzzy Information Granularity and Their Representations

Yutaka Hata†
†Department of Computer Engineering,
Himeji Institute of Technology
2167, Shosha, Himeji,
671-2201, JAPAN
hata@comp.eng.himeji-tech.ac.jp

Masao Mukaidono‡
‡Department of Computer Science,
Meiji University
1-1-1 Higashimita, Tamaku, Kawasaki,
214-8571, JAPAN
masao@cs.meiji.ac.jp

Abstract

This paper describes some classes of fuzzy information granularity and their representation methods. Fuzzy information granularity introduced by Zadeh is that " granularity relates to clumpiness of structure, while granulation refers to partitioning an object into a collection of granules, with a granule being a clump of objects(points) drawn together by indistinguishability, similarity, proximity, or functionality." In this paper we show three classes of granularity structures, which are called Kleene class, Lukasiewicz class and probabilisticlike class. Their meaning and representation methods are discussed. Their examples are also demonstrated. This paper would be a basis to research on the representation of fuzzy information granularity.

1. Introduction

A concept that plays a pivotal role in fuzzy logic[1]-[4] is that of fuzzy information granulation[1]. In crisp information granulation, the granules are crisp, while in fuzzy information granulation the granules are fuzzy. Granularity relates to clumpiness of structure, while granulation involves decomposition of the whole into the parts, i.e., an object into a collection of granules. The granule is a clump of objects drawn together by indistinguishability, similarity, proximity, or functionality. Moreover, each fuzzy attribute is associated with a set of fuzzy values. Our departure point of this study is that for a fuzzy granularity we will clarify the structure of the clumpiness by representing it by fuzzy logic operators. By exploring it, we would be form the basis of understanding the clumpiness, classify it and then represent it.

This paper describes three classes of fuzzy information granularity (FIG for short) and their representation methods. First, the three classes are called by: Kleene[5] class, which can be expressed by maximum, minimum and not operations; Lukasiewicz class, which can be expressed by bold union, bold intersection[3] and not; Probability class, which can be expressed by probabilisticlike sum, probabilisticlike product and not. Second, as a representative of Kleene class, we introduce fuzzy logic functions[6]-[10], which is a model Kleene algebra[5]-[12]. The granularity expressed by fuzzy logic functions are shown and then an application to image granulation is demonstrated. Third, Lukasiewicz class is introduced as the class that can express FIG including information specified by standard arithmetic sum and subtraction calculation. As the application, we show the granulation of human brain MR image[14][15][16]. Finally, Probability class is introduced as the class of FIG including probabilistic events. This paper is organized below. Section 2 defines three classes mentioned above. Section 3 describes Kleene class of FIG. First, fuzzy logic function is defined as the representative, and then is applied to granulate a given image. Section 3 describes Lukasiewicz class and the effectiveness. Section 4 shows the example of Probability class. Section 5 is the conclusions.

2. Preliminaries

As far as fuzzy information granulation[1] introduced by Zadeh, the granules are fuzzy. Granularity relates to clumpiness of structure, while granulation involves decomposition of the whole into the parts. Figure 1 shows the structure of granulation shown by Zadeh. Thus, granulation can be decomposed into information granulation and action granulation, the information granulation can be decomposed into fuzzy granulation and crisp granulation. Consider a granularity of an object, Zadeh shows the structure shown in Figure 2. On focusing on the fuzzy information granularity, we have to clarify the granulation method, representation method of the granularity. In this paper, we will consider fuzzy information granularity, especially, granularity representation method. Crisp information granularity may have the same representation method. First, we consider the following three classes based on logic operators.

Let V = [0, 1] be the interval between 0 and 1, and let X = (x1, x2, ... , xn) be a set of n variables where xi takes on values from U. A function f(X) is a mapping $f : U^n \rightarrow U^1$. f(X) is said to be an n-variable fuzzy function. In this paper, the following three sets of fuzzy logical operators are considered.

(1) <MIN, MAX. NOT> : **Kleene**[5] **class**

1) MIN : $f(x1, x2) = x1 \wedge x2$ = minimum (x1,x2),

2) MAX : $f(x1, x2) = x1 \vee x2$ = maximum (x1, x2).

3) NOT: $f(x1) = \overline{x1} = 1\text{-}x1$

0-7695-0161-3/99 $10.00 © 1999 IEEE

(2) <BOLD INTERSECTION, UNION, NOT>[3] : **Lukasiewicz class**

1)BOLD INTERSECTION : f(x1, x2) = x1•x2 = maximum (x1+ x2-1, 0),where "+" is arithmetic sum,

2) BOLD UNION: f(x1, x2) = x1 + x2 = minimum (x1+ x2, 1), where "+" is arithmetic sum,

3) NOT: f(x1) = $\overline{x1}$=1-x1

Bold intersection and union are also refered as bounded product and sum, respectively. Since Lukasiewicz implication A→ B (= minimum ((1-A) + B, 1)) can be expressed by $\overline{A}$+B, we call this Lukasiewicz class

(3) <PROBABILISTICLIKE PRODUCT, SUM, NOT>:**Probability class**

1) PROBABILISTICLIKE PRODUCT :
f(x1, x2) = x1 × x2

2) PROBABILISTICLIKE SUM :
f(x1, x2) = x1 ∇ x2 = x1 + x2 - x1 × x2

3) NOT: f(x1) = $\overline{x1}$=1-x1

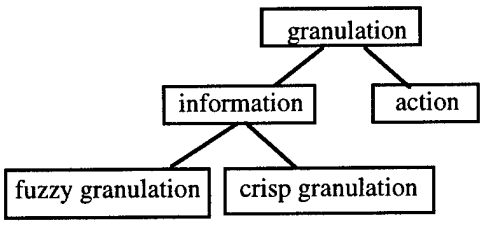


Figure 1 A granulation relation.

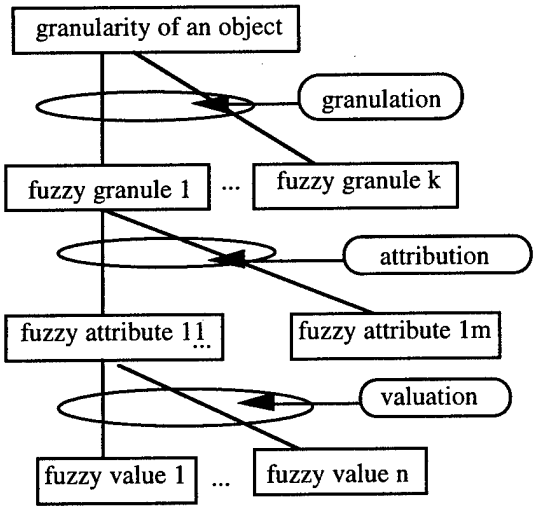


Figure 2 Granularity, attribute and values.

In the following sections, we are considering these fuzzy information granularity (FIG for short) representation, and then show their representation methods and their application area.

3. Representation of FIG on Kleene Class

This section describes FIG representation on Kleene class. First we show the definition of fuzzy logic functions[6]-[10], which is a model of Kleene algebra[5][11][12].

[Definition 1] Fuzzy logic formulas are defined as follows:
(1) A variable Xi is a fuzzy logic formula.
(2) If Hi and Hj are fuzzy logic formulas, $\overline{Hi}$ Hi∨Hj and Hi∧Hj are fuzzy logic formulas.
(3) The only formulas are those given by rules (1) and (2).
Fuzzy logic functions are defined as the fuzzy functions that can be represented by the fuzzy logic formulas.

All fuzzy logic functions can be represented by both sum of product and product of sum, because Idempotency (X∧X=X, X∨X=X), Distributively X1∧(X2∨X3) = X1∧X2∨X1∧X3, X1∨X2∧X3 =(X1∨X2)∧(X1∨X3) and so on are approved in fuzzy logic. But complementation (X∧$\overline{X}$=0, X∨$\overline{X}$=1) does not hold, whereas the Kleene's law (X∧$\overline{X}$ ≤ Y∨$\overline{Y}$) does hold. So the term with both Xi and $\overline{Xi}$ can remain. Such product terms are called complementary product terms, and alterms are called complementary alterms. The other product terms called simple product terms, and alterms are called simple alterms.

[Example 1] Let the following terms be the terms of three-variable fuzzy logic functions. Simple product terms, simple alterms, complimentary product terms, complementary alterms, minterms and maxterms are those below, respectively,

simple product terms: X1, X1∧X2, X1∧$\overline{X3}$
simple alterms: X2∨X3, X1∨$\overline{X2}$
complementary product terms: X1∧$\overline{X1}$, X1∧X2∧$\overline{X2}$
complementary alterms: X3∨$\overline{X3}$, X1∨$\overline{X1}$∨X2

Any fuzzy logic function F can be expressed by the following form,

$F = \alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_n$, (1)

where α_i is one of simple product term or complementary product term.

$F = \beta_1 \wedge \beta_2 \wedge \ldots \wedge \beta_m$, (2)

where β_i is one of simple alterm or complementary alterm.

Considering the functions F as the fuzzy information α_i (i=1, ...,n) and β_i (i=1, ..., m) are the fuzzy granules. Here, the method to identify and then represent fuzzy logic formula for a given fuzzy function, see References[11] and [12]

[Example 2] We consider the granularity of the image of fuzzy function F in Figure 3. In it, more white pixel includes higher value in [0, 1]; The upper illustrations

show the shapes of projection by putting a light from direction of the axis of X2 in F, G1, G2, X1 and $\overline{X1}$, X2=1/2. When we represent the image by fuzzy logic function, F can be expressed by $(X1 \wedge \overline{X1}) \wedge (X2 \wedge \overline{X2})$. In the similar way, $G1 = X1 \wedge \overline{X1}$ and $G2 = X2 \wedge \overline{X2}$. Clearly minimum(G1, G2)=F. The image is thus granulated to fuzzy functions G1 and G2.

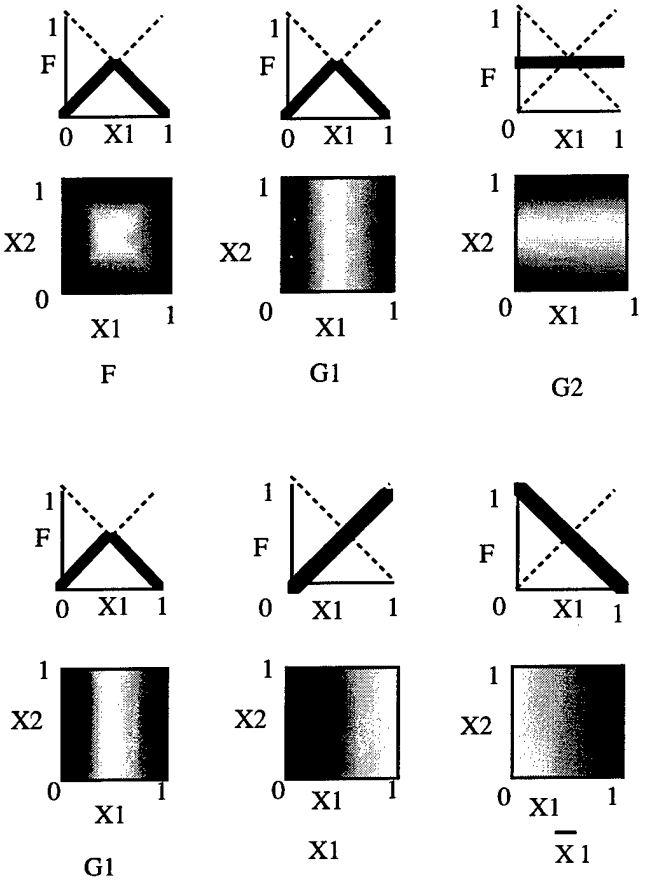


Figure 3 An illustration on FIG of fuzzy logic function

We define the complete granule set below.

[Definition 2] Suppose a granularity of an object, the granule set that can completely express the granularity is called as complete granule set with respect to the object.

Considering Equations (1) and (2), the sets $\{\alpha_1, \alpha_2, ..., \alpha_n\}$ and $\{\beta_1, \beta_2, ..., \beta_n\}$ are complete granule sets with respect to F. When $\alpha_i = \gamma_1 \wedge \gamma_2 \wedge ... \wedge \gamma_k$, the set $\{\gamma_1, \gamma_2, ..., \gamma_k\}$ is a complete granule set with respect to α_i. We can obtain the same discussion with respect to β_i.

[Example 3] Consider the fuzzy logic function $F = (X1 \wedge \overline{X1}) \wedge (X2 \wedge \overline{X2})$ of Example 2. First granulation of F can derive two granules $(X1 \wedge \overline{X1})$ and $(X2 \wedge \overline{X2})$, Moreover, second granulation of $(X1 \wedge \overline{X1})$ can derive X1 and $\overline{X1}$, and that of $(X2 \wedge \overline{X2})$ can derive X2 and $\overline{X2}$. Here the set of $(X1 \wedge \overline{X1})$ and $(X2 \wedge \overline{X2})$ is complete granule set. The set of X1 and $\overline{X1}$ is complete granule set with respect to $(X1 \wedge \overline{X1})$, but is not complete granule set with respect to F. However, the set of X1, $\overline{X1}$, X2 and $\overline{X2}$ is a complete granule set with respect to F.

Thus, if we can represent the given object by fuzzy logic functions, we can easily represent and then granulate it to granules.

4. Representation of FIG on Lukasiewicz Class

The class which can be expressed by bold intersection, union and NOT is considered in this section. The class satisfies commutativity, associativity, identity, De Morgan's laws, excluded-middle laws and involution. But idempotency, distributivity and absorption are no longer. Therefore, it is immediate that, with the operations, the subset is not a lattice but a complemented non distributive structure. When {0, 1}, the set is a complemented distributive lattice, i.e., a Boolean algebra.

Many of pattern recognition and Sensor-based detection system problems may be classified into this class, because bold union means arithmetic summation under the assumption that its value is not beyond a limitation, where we denote the limitation by "1". If we consider a sensor which detects some signal from an object, the sensor has physically detectable limitation exactly. Considering the limitation by the logic value "1", the class would widely be applicable to real problems.

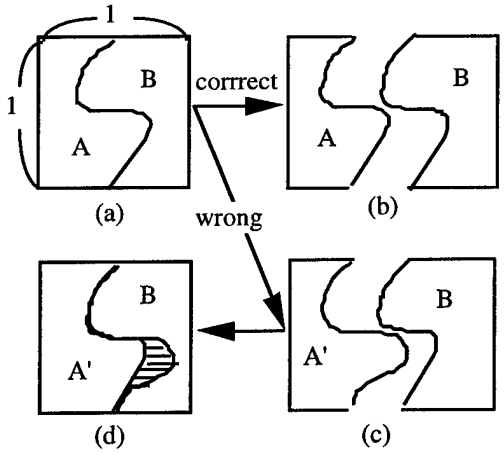


Figure 4 An illustration of granulation on Lukasiewicz class.

Suppose a granulation of the square in Figure 4(a). Since the whole area is 1, the limitation is set as "1". If

we correctly granulate it to the parts A and B, then, T(A) + T(B)=1 and T(A) • T(B)=0 hold. If we granulate to Part A' being bigger than Part A, and Part B as shown in Figure 4(c), then T(A') + T(B)=1 and T(A') • T(B) = the lined area in Figure 4(d). If we granulate to Part A" being smaller than Part A, and Part B, then T(A") + T(B) = lacked area and T(A") • T(B)=0. The following relation thus holds.

[Relation 1: Lukasiewicz Class] Consider the granulation of the whole into parts x1, x2, ..., xn. We normalized the values xi (i=1, ..., n) by the whole values, i.e., T(xi) = values(xi) / value(the whole), then upper bound of T(xi) becomes 1.

If $\sum_{i=1}^{n} +T(xi) < 1$ then $(1 - \sum_{i=1}^{n} +T(xi))$ non granulated granules exist,

else if $\prod_{i=1}^{n} \bullet T(xi) > 0$ then $\prod_{i=1}^{n} \bullet T(xi)$ over granules exist.

else if $\sum_{i=1}^{n} +T(xi)=1$ and $\prod_{i=1}^{n} \bullet T(xi)=0$ then correctly granulated: the whole into the parts x1, x2, ..., xn. The set of x1, x2, ..., xn is a complete granule set with respect to the whole.

[Example 4] We consider the granulation of human brain MR image. We detected the human brain MR image by a MRI device. All the sliced images are generated from neck to top of brain as shown in Figure 5(a). Figure 5(b) shows the granulated brain area. The whole brain(white matter and gray matter) is composed by left cerebral hemisphere(LCH), right cerebral hemisphere(RCH), cerebellum(CB) and a part of brain stem(BS). We set the limitation of bold union as the volume of whole brain, i.e., all volumes of LCH, RCH, CB and BS are divided by the volume of whole brain, and then denote T(A) = Volume(A)/Volume(whole brain). Then, T(LCH) + T(RCH) + T(CB) + T(the part of BS) must be 1, and T(LCH) • T(RCH) • T(CB) • T(the part of BS) must be 0. Otherwise we determine that the granulation is wrong.

4. Representation of FIG on Probability Class

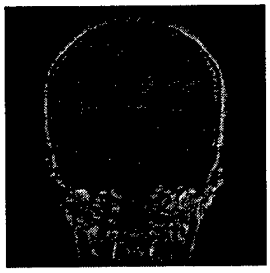
In this section, we will consider FIG on Probability class.

For Probability P(A) and P(B) for possible events A and B,

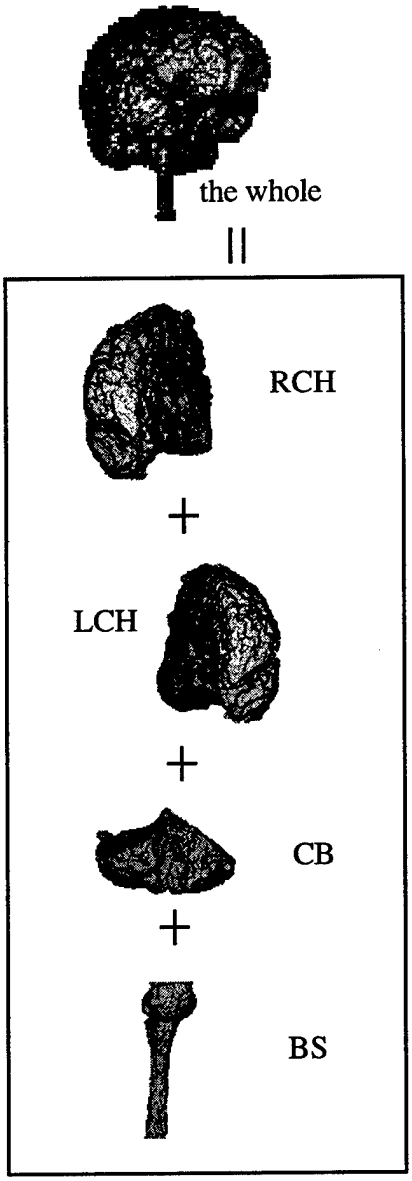
P(A or B) = P(A) + P(B) - P(A and B),

P(A and B) = P(A) × P(B/A) = P(B) × P(A/B), where P(A/B) means the conditional probability of A occurring, given that B has occurred.

In our probability class, on considering P(A/B)=P(A) and P(B/A) = P(B), we derive "∇" and "×". We therefore call these operators "∇" and "×" probabilisticlike sum and product, respectively. Under these operations, "∇", "×" and "NOT", the set is only a pseudocomplemented non



(a) A raw image of human brain MR image.



(b) Human brain MR image granulation of Example 4.

Figure 5 An example on Lukasiewicz class.

distribute structure. More particularly, "∇" and "$\times$" satisfy only commutativity, associativity, identity, De Morgan's laws and $A \times 0 = 0$, $A \nabla 1 = 1$.

[Example 5] **Problem:** Suppose that we have a research meeting tomorrow. Please estimate whether the probability of the number of attendees is up or down, compared to the number of attendees in last year ?

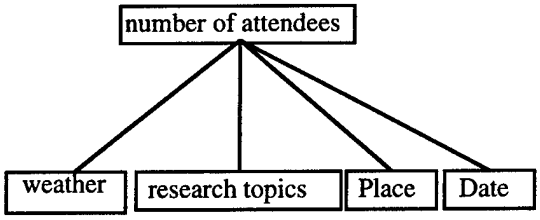


Figure 6 A granularity structure of Example 5: Probability class.

Table 1 : Data table of Example 5

	last year	this year
attendees #	30 (total researcher # :about 50)	?
the weather	cloudy	maybe, fine
research topics	fuzzy logic(FL) and neural networks(NN)	Chaos(CA) and Evolutionary computing(EC)
Date	first Thursday, November	first Thursday, November
Place	Himeji	Himeji

The granularity and an estimation: Considering the granularity of the number of attendees, we can have a FIG structure as shown in Figure 6.

<granule of the weather> : If it is rainy, the attendees number may be fewer, compared to cloudy day or fine day. In this problem, we do not have to take care to the weather granule.

<granule of the research topics>: Assume that we have the following information about 50 researchers. $P_T(FL)=0.7$, $P_T(NN)=0.4$, $P_T(CA)=0.3$ and $P_T(EC)=0.5$, where $P_T(x)$ denotes a probabilistic measure (percentage) of the number of researchers who are interested in topics x by analyzing past questionnaire. Then,

$P_T(FL \text{ or } NN) = P_T(FL) \nabla P_T(NN) = 0.7 + 0.4 - 0.7 \times 0.4 = 0.82$,

$P_T(CA \text{ or } EC) = P_T(CA) \nabla P_T(EC) = 0.3 + 0.5 - 0.3 \times 0.5 = 0.65$,

Thus, $P_T(FL \text{ or } NN) > P_T(CA \text{ or } EC)$, the fuzzy vales of attribute of the granule of the research topics (CA and EC) are smaller than that of (FL and NN).

<granule of the place and date> : The place and date in tomorrow meeting are as same as those in last year.

Considering the probability of the number of attendees according to their granules, the degree will be smaller, i.e., we estimate that the number of attendees is smaller than the number of attendees in last year.

Thus, the probability class is also a meaningful class in FIG. This class can express the structure which includes events of probabilisticlike properties. In Example 5, the calculation on granule of research topics seems to be right. However, considering the granularity composed of granules, weather, research topics, place and date, the FIG class is not uniquely determined, i.e., maybe it is classified into either Lukasiewicz class or probabilistic class. Moreover, in real world problem we have the combination/fusion classes of the two or three classes.

5. Conclusions

This paper describes three classes of fuzzy information granularity and their representation methods: Kleene class, Lukasiewicz class and probability class. As a representative in Kleene class, fuzzy logic functions are shown, and then the application to the image granulation is demonstrated. Lukasiewicz class can express FIG including information specified by standard arithmetic sum and substation calculation. As the application, we show the granulation of human brain MR image. Probabilistic class is introduced as the class of FIG including probabilistic events. In the real world problem, there are many combination/fusion classes of them. However, this paper would be a basis to clarify the representation of fuzzy information granularity.

Acknowledgment

Research supported in part by Tateishi Science and Technology Foundation, Ishikawa Hospital Grant and the BISC Program of UC Berkeley.

References

[1] L. A. Zadeh : "Toward a theory of fuzzy information granulation and its centrality in human reasoning and fuzzy logic", Fuzzy Sets and Systems, 90, pp.111-127 (1997)

[2] L. A. Zadeh: Fuzzy sets and Applications, John Wiley and Sons, Inc. (1987).

[3] D. Dubois and H. Prade: Fuzzy Sets and Systems: Theory and Applications, Academic Press, 1980

[4] W. Pedrycz: "Fuzzy Control And Fuzzy Systems, "Research Studies Press Ltd. (1993).

[5] S.C.Kleene, Introduction to metamathematics, Amsterdam North-Holland, 1952

[6] P.N.Marinos," Fuzzy logic and its application to switching systems", IEEE. Trans. Comput., vol. C-18, pp.343-348 (1969).

[7] A.Kandel, "On minimization of fuzzy functions", IEEE. Trans Comput., vol. C-22, pp. 826-832, (1973).

[8] R.C.T.Lee and C-L. Chang, "Some properties of fuzzy logic", Inf. and Control, vol. 19, pp.417-431 (1971).

[9] M.Mukaidono, "On some properties of fuzzy logic", Trans. IECE. Japan, vol.58-D, pp.150-157 (1975).
[10] M.Mukaidono,"An algebraic structure of fuzzy logic and its minimal and irredundunt form", Syst. Comput. Contr., vol. 3 pp.60-68 (1975).
[11] Y.Hata, K.Nakashima and K.Yamato "Some Fundamental Properties of Multiple-Valued Kleenean Functions and Determination of Their Logical Formulas" IEEE Trans. of Comp., Vol. 42, No. 8, pp.950-961 (1993)
[12] N.Takagi, H. Kikuchi, K. Nakashima and M.Mukaidono " Identification of Incompletely Specified Multiple-Valued Kleenean Functions" IEEE Trans. of System, Man and Cybernetics-PartA, Vol. 28, No.5, pp.637-647 (1998)
[13] J. C. Ross "The Image Processing Handbook. second editon", CRC Press, 1995
[14] S. Kobashi, N. Kamiura and Y. Hata, "Fuzzy Information Granulation on Segmentation of Human Brain MR Images," Journal of Japan Society for Fuzzy Theory and Systems, vol. 10, no. 1, pp. 117-125, Feb. 1998.
[15] Y. Hata, S. Kobashi, N. Kamiura and M. Ishikawa, "Fuzzy Logic Approach to 3D Magnetic Resonance Image Segmentation," Information Processing in Medical Imaging, Lecture Notes in Comp. Sci., Vol. 1230, pp. 387-392, Jun. 1997.
[16] Y. Hata, S. Hirano and N. Kamiura, "Medical Image Granulation by Fuzzy Inference," Proc. of the 17th Annual Meeting of the North American Fuzzy Information Processing Society - NAFIPS, pp. 188-192, Aug. 1998.

From a Fuzzy Flip-Flop to a MVL Flip-Flop

LP Maguire, TM McGinnity, LJ McDaid

Intelligent Systems Engineering Laboratory,
Faculty of Engineering, Magee College,
University of Ulster, Northland Rd., Derry,
Northern Ireland, UK, BT48 7JL

E-mail: lp.maguire@ulst.ac.uk

Abstract

The paper presents a circuit description of a MVL flip-flop which is implemented in MOS technology. The circuit has its origins in a bipolar implementation of a fuzzy flip-flop. The proposed circuit not only simplifies the original bipolar design but it offers considerable potential for a VLSI implementation. Simulation of the circuit using HSpice indicates that it can also be interpreted as a multiple-valued logic flip-flop for any radix. This provides a basic building block for the increasing developments in multiple-valued VLSI circuit design.

1. Introduction

Multiple-valued logic (MVL) is making an increasing impact on VLSI circuit design [1]. The theoretical advantages of MVL for pinout and interconnect-limited IC design have been well established and widely acknowledged, as signals are allowed to assume more than two states. A number of hybrid primitives [2] have been designed for the hardware implementation of multiple valued logic, eg the T-gate [3] and the U-gate [4]. More recent developments have introduced designs for a multiple-valued logic flip flop [5-10].

More recently there has been a complementary trend in the development of circuit realisations of the fuzzy inference process [11]. An important contribution in this area was the introduction of the concept of a fuzzy flip-flop [12-16], which is an extension of the conventional binary flip-flop. This research introduced the flip-flop as a basic building block for the realisation of a fuzzy processor.

There are obvious similarities between fuzzy logic and multiple-valued logic. In particular, the authors believe that one method of exploiting these similarities is to use fuzzy sets to represent each level in an MVL system. This would have the effect of removing the distinct boundaries associated with conventional bivalent sets and allow the representation of each level to overlap.

The major benefit of such an approach is that a MVL system could then readily accommodate differing numbers of levels within the restrictions of the conventional power rails and that the noise margins required between levels would be facilitated by the approximate nature of the fuzzy based description. This approach offers considerable potential for VLSI implementations which are currently motivated by lower power designs. This feature provides the main motivation for the work presented in this paper.

The paper begins with the description of the bipolar JK fuzzy flip-flop originally proposed by Hirota [14]. The MOS implementation of this circuit is then presented which not only simplifies the original bipolar design but it also offers considerable potential for a VLSI implementation. The main contribution of this work however, demonstrates how this circuit can also be interpreted as a multiple-valued logic flip-flop for any radix.

The paper is divided into a number of sections. Section 2 provides an introduction to the concept of a fuzzy flip-flop and describes the MOS implementation of this circuit. Section 3 then provides a multiple-valued logic interpretation of the fuzzy logic flip-flop for any radix. Finally section 4 concludes the paper with a

0-7695-0161-3/99 $10.00 © 1999 IEEE

discussion of the results and an outline of the conclusions drawn from the work.

2. The fuzzy JK flip-flop

Hirota defined the set and reset type characteristic equations of a binary JK flip flop as a fuzzy flip-flop using the concepts of fuzzy negation, t-norm and s-norm [14]. The characteristic equations of the reset and set forms of the fuzzy JK flip-flop are defined by the following equations:

$$Q_S(k+1) = (J \; s \; Q) \; t \; (K^n \; s \; Q^n) \qquad (1)$$

$$Q_R(k+1) = (J \; t \; Q^n) \; s \; (K^n \; t \; Q) \qquad (2)$$

where Q_S and Q_R are the set and reset forms respectively; t = t-norm operator; s= s-norm operator; and the superscript n is used to represent complementation.

There are a number of operators which can be used to realise these functions but when Min, Max and 1- are employed for the t-norm, s-norm and complementation operators respectively; this simplifies equations 1 and 2 to the following:

$$Q_R(k+1) = \{J \; \text{Min} \; (1-Q)\} \; \text{Max} \; \{(1-K) \; \text{Min} \; Q\} \qquad (3)$$

$$Q_S(k+1) = \{J \; \text{Max} \; Q\} \; \text{Min} \; \{(1 - K) \; \text{Max} \; (1 - Q)\} \qquad (4)$$

Hirota described how equations 3 and 4 can be reduced to provide the fundamental equation of the Min-Max type fuzzy flip-flop as defined by equation 5

$$Q(t+1)= \{J \; \text{Max} \; (1-K)\} \; \text{Min} \; (J \; \text{Max} \; Q) \; \text{Min} \; \{(1-K) \; \text{Max} \; (1-Q)\} \qquad (5)$$

This defining equation was implemented using bipolar technology and realised using discrete components (using 38 transistors, 44 resistors, 10 diodes and 2 capacitors). A complete circuit schematic and a description of the results obtained are provided in the original paper [14].

This bipolar design was modified by the authors, to provide a MOS implementation of this fuzzy flip-flop circuit (using 62 transistors). The operation of this circuit as a fuzzy flip-flop was confirmed by simulation using HSpice. A block diagram of the complete circuit of the flip-flop is provided in Figure 1 at the end of the paper. In this modular representation, the circuit is broken down into a number of functional blocks as described by the following:

Min: Circuit required to implement the Min operation

Max: Circuit required to implement the Max operation

Negate: Circuit required to implement the complement operation

Trans: Circuit required to implement a transmission gate

Inv: Circuit required to implement an inverter

The circuit realisation for the Min, Max and Negate blocks (which have been copied from the full working schematic) are provided in Figures 2, 3 and 4 respectively. In these circuits all the transistors are depletion mode devices.

Figure 2 describes the Min circuit where Q3 and Q5 are configured as passive pull-up or pull-down resistors. Both the input voltages (I/P1 and I/P2) and the output voltage (O/P) can take values in the range VEE to VCC. Consider the case where I/P1 is more negative than I/P2 and assume that both Q1 and Q2 are operating above pinch off. This assumption will always be valid if the gate/source voltage of both Q1 and Q2 is small. If the transconductance of both Q1 and Q2 is maintained large while that associated with Q3 is maintained small, then the voltage on the gate of Q4 will track with the lowest input voltage. With Q4 and Q5 configured as a voltage follower, the output voltage (O/P) will follow that gate voltage of Q4. Therefore, the output voltage will follow the lower of the two inputs.

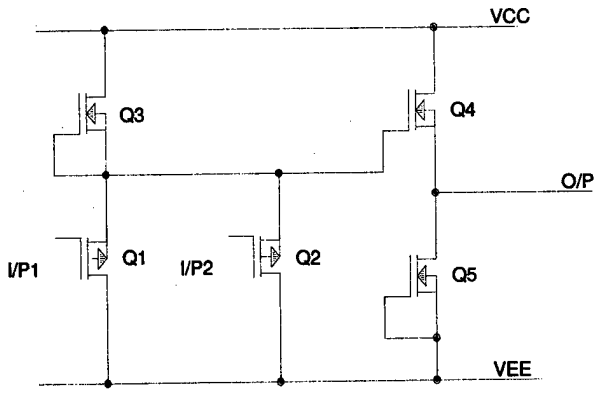


Fig 2 : Schematic for the Min circuit block

The Max circuit presented in Figure 3 operates in a similar fashion, where the voltage at the gate of Q5 will follow either the gate voltage of Q1 or Q2, whichever is the higher. The output circuit in Figure 3 is a voltage follower and therefore the output voltage (O/P) will

follow the gate voltage of Q5 and hence the larger of the two input voltages.

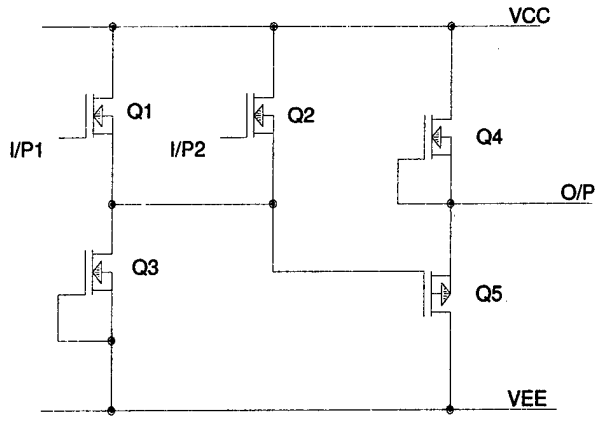


Fig 3 : Schematic for the Max circuit block

The circuit in Figure 4 consists of a voltage follower, a constant current source and a differential amplifier with an active load. When the input voltage (Vin) increases, the gate voltage, and hence drain current, of Q9 increases resulting in a proportional decrease in the drain current in Q8. In addition, because Q6 and Q7 operate in a current mirror configuration, the drain current of Q7 decreases. Hence, a current flows in the output resistor, R3, such that the output voltage Vo falls.

However, if the input voltage falls, then the gate voltage, and consequently drain current, of Q9 falls, causing a proportional increase in the drain current associated with Q8. Owing to the action of the current mirror, the drain current of Q7 increases causing a current to flow in the output resistor R3. The direction of this current is such that the output voltage increases. Essentially, the circuit operates as an inverter and in the design process R1, R2 and R3 can be chosen to yield a transfer function obeying

$$Vo = Vcc - Vin \qquad (6)$$

which is equivalent to the Negate operation for the logic levels.

The resistors (R1-R3) can be implemented as MOS transistors and are used here simply to aid understanding of the functionality of the circuit.

The MOS circuit not only reduced the component count but it is more suitable for VLSI implementation. The circuit was simulated using HSpice providing results which agreed with the fundamental equation as defined by equation 5. The next section describes how these results can be interpreted for a MVL flip-flop. Ongoing work, within the research group, is refining the design and realising the circuit in hardware as a full-custom VLSI design.

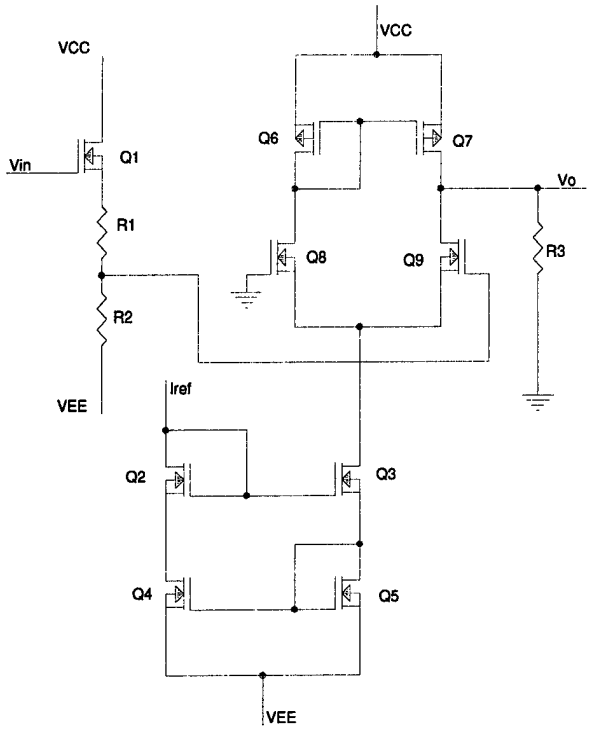


Fig 4 : Schematic for the Negate circuit block

3. The multiple-valued logic interpretation.

The flip-flop circuit was simulated using the HSpice simulation package. Inputs were confined to the range 0-5V and the results confirmed those reported in the original paper [14]. In this work the circuit was interpreted as a MVL flip-flop and the authors partitioned the input space into a number of fuzzy sets for each of the logical inputs. This strategy used triangular fuzzy sets with unity partition as shown in Figs 5, 6 and 7 below. For example, in Fig 6 a logic 1 is interpreted as having full membership at a voltage level of 2.5V and partial membership extending to the 0V and 5V limits. Similarly all the other fuzzy sets have the logical level represented by a central voltage with decreasing membership at all other voltages.

This representation was used as a reference to interpret the results of the simulations during which the inputs were initialised at that voltage corresponding to the maximum set membership. In practice the output voltage closely matched one of these central voltages with a

maximum deviation of ±0.5V. Ongoing work within the group is investigating the effect of varying each input within its set description and noting the tolerance of the output.

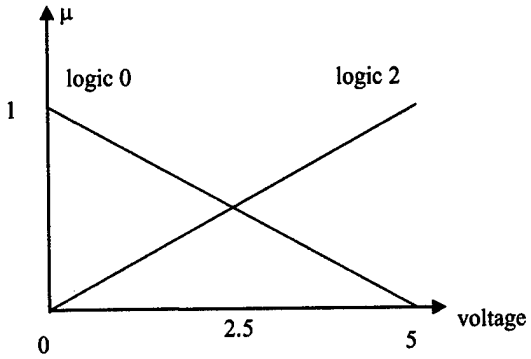


Fig 5: Logic level assignment when radix =2

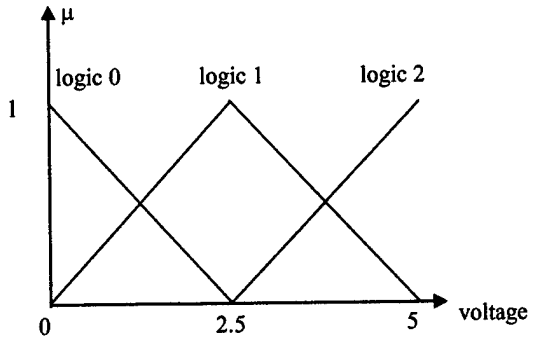


Fig 6: Logic level assignment when radix =3

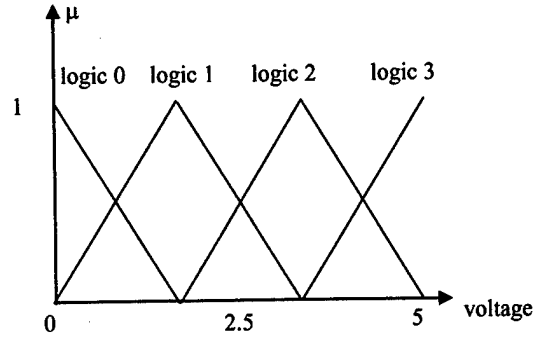


Fig 7: Logic level assignment when radix =4

The results of the Spice simulation are summarised in Tables 1, 2 and 3. These results document the truth tables of the flip-flop circuit using radices of 2, 3 and 4 respectively.

K \ J	0	1
0	Q	1
1	0	Q*

Table 1: Derived truth-table for MVL flip-flop when radix=2

K \ J	0	1	2
0	Q	Max(Q,1)	2
1	Min(Q,1)	1	Max(Q*,1)
2	0	Min(Q*,1)	Q*

Table 2: Derived truth-table for MVL flip-flop when radix=3

K \ J	0	1	2	3
0	Q	Max(Q,1)	Max(Q,2)	3
1	Min(Q,2)	Max[Min(Q*,1), Min(Q,2)]	2	Max(Q*,2)
2	Min(Q,1)	1	Max[Min(Q,1), Min(Q*,2)]	Max(Q*,1)
3	0	Min(Q*,1)	Min(Q*,2)	Q*

Table 3: Derived truth-table for MVL flip-flop when radix=4

where the complement operator is defined as:

$$Q^* = (p - Q)$$

and p is given by:

$$p = R-1$$

where R is the radix.

The terms in the truth table are represented using the Min and Max operators and some examples of these operations are illustrated in Table 4 using a radix of 4.

Q	Min (Q,1)	Max(Q,2)	Max[Min(Q*,1), Min(Q,2)]
0	0	2	1
1	1	2	1
2	1	2	2
3	1	3	2

Table 4: Examples of terms in the MVL flip-flop truth table (when radix=4)

Examining Tables 1, 2 and 3 reveal that there is clearly a pattern emerging as the next state of the output is dependent on a combination of the present state and a fixed value. Indeed there is a degree of symmetry in the truth tables. For example if you consider the truth table for radix 3 presented in Table 2 and use DeMorgan's theorem one can show that the following relationship applies:

$$Min(Q,1) = [Max(Q^*,1)]^*$$

These truth tables differ from previously reported MVL designs, as they are more dependent on the previous value of the output. However, there is clearly a pattern emerging as the output can be set to a given value, complemented, or made a function of the previous output. This pattern can be used to predict the truth table for any chosen radix; this was confirmed by simulation.

4. Discussion and Conclusions.

The research presented in this paper describes the MOS implementation of a fuzzy flip-flop which is appropriate for a VLSI implementation. The simulation results demonstrated the successful operation of the circuit as a fuzzy flip-flop.

The main contribution of the work is the demonstration that this fuzzy flip-flop can also be interpreted as a MVL flip-flop. The authors described how the logic levels were represented using fuzzy sets and simulation results were presented to confirm the circuit's operation. The results indicate a clear pattern emerging in the truth tables for any chosen radix.

This paper reports on the initial investigations into the use of fuzzy logic and fuzzy set descriptors in the interpretation of logic levels for MVL. Such an interpretation offers considerable potential as the application of MVL logic circuit is becoming increasingly important in digital signal processing and computing applications. The current research has not yet exploited the noise immunity capabilities of the fuzzy reasoning process in this interpretation but it is the authors opinion that this will be employed when such sub-circuits are employed in a complete design. This area of research is being currently investigated by staff of the Intelligent Systems Engineering Laboratory.

References

[1] K Wayne Current: "Current-mode CMOS multiple-valued logic circuits", IEEE Journal of Solid-State Circuits, pp.95-107, Vol. 29, No. 2, 1994.

[2] T. Yamakawa: "CMOS multi valued circuits in hybrid mode", Proc. of the 15th IEEE Int Symposium on MVL, pp. 144-151, 1985.

[3] B.P. Chew, H.T. Mouftah: "On the design of CMOS ternary logic circuits using T-gates", Int. Journal of Electronics, Vol. 63, pp. 229-239, 1987.

[4] J.A.C. Webb, S.J. Laverty: "Extension of the T-gate concept to a hybrid U-gate architecture for ternary and higher order logic systems", Electronics Letters, Vol. 26, No. 19, pp. 1629-1630, 1990.

[5] N. Zhuang, H. Wu: "Novel ternary JKL flip flop", Electronics Letters, Vol. 26 , No. 15, pp. 1145-1146, 1990.

[6] J.A.C. Webb, S.M.N. Forbes, J. Wilson, S.J. Laverty: "Hybrid higher radix JK flip flop sequencer with ASIC implementation potential", Electronics Letters, Vol. 27, No. 21, pp. 1933-1935, 1991.

[7] X. Wu, X. Chen: "The research of ternary D-type triggered Flip Flops", Texue Tongbao (Science Bulletin), Vol. 15, pp. 1060-1064, 1987.

[8] J.I. Acha, J.L. Huertas: "General Excitation table for a JK multistable", Electronics Letters, Vol. 11, p. 624, 1975

[9] Q Chen:"A flip-flop with complete functions", Chinese Journal of Computers, (In Chinese) Vol 15, No. 6, pp. 479-481, 1992

[10] S Karasawa; K Yamanouchi: "Design and examination of a multiple-valued flip-flop circuit with stair shaped I-V curved device as a coupling element", Proc 23rd IEEE Int Symposium on MVL, pp. 152-157, 1993.

[11] T. Yamakawa: "A fuzzy inference engine in non-linear analog mode and its application to a fuzzy logic control" IEEE Trans. on Neural Networks, pp.496-522, Vol. 4, No. 3, 1993.

[12] Y Mori, M Mukaidono: "Fuzzy Logic simulator for sequential circuits expressed by logical formulas", Proc 1st Asian Fuzzy Systems Symposium, Singapore, pp538-543, Nov 1993.

[13] Y Mori, M Mukaidono: "Fuzzy Flip-flops expressible with logical expression and their properties", Proc Korean-Japan Joint Conf on Fuzzy Systems and Engineering, pp165-168, 1992.

[14] K. Hirota, K. Ozawa: "Concept of a fuzzy flip flop", Preprints of Second IFSA Congress (Tokyo, July'87), pp. 556-559, 1987.

[15] K. Hirota, K. Ozawa: "Concept of a fuzzy flip flop", IEEE Trans. Systems Man and Cybernetics, Vol. 19, No. 5, pp 980-997, 1989.

[16] N. Ikoma, K. Ozawa, K. Hirota and W. Pedrycz: "Fuzzy sequential circuit based on weight-added fuzzy flip flop", Proc. IEEE Int. Conf. on Fuzzy Systems, pp. 291-296, 1994.

Acknowledgements

The authors acknowledge the assistance of Ms Zhen Sun for providing an English translation for reference [9].

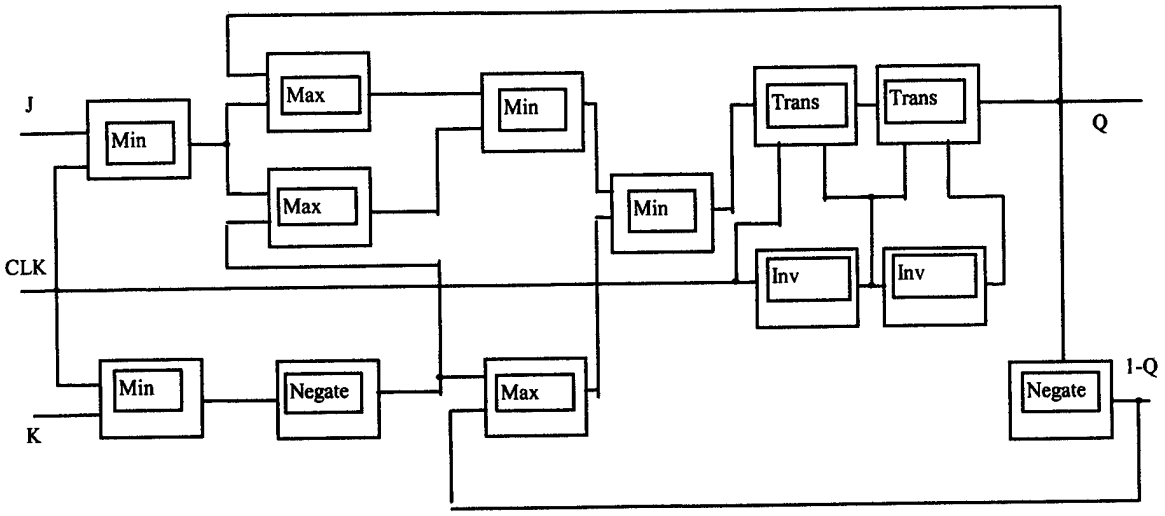
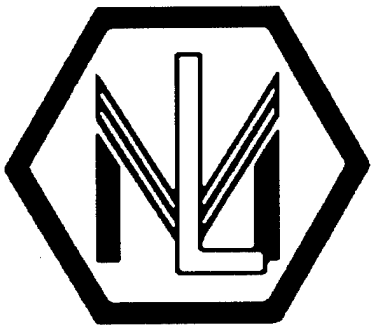


Fig. 1 : Block diagram description of the MOS implementation of the fuzzy flip-flop

AUTHOR INDEX



Author Index

Abd-El-Barr, M. 160, 262
Adams, K. 220
Ahmadi, M. 186
Al-Sherif, M. 262
Aoki, T. 200
Ayari, A. 142
Baba, T. 2
Babu, H. Md. H. 166
Basin, D. 142
Beckert, B. 248
Brzozowski, J. 66
Campbell, J. 220
Debnath, D. 99
Dubrova, E. 92
Dueck, G. 118
Fernandes, H. 160
Fraser, B. 118
Freitag, K. 194
Friedrich, S. 142
Fugère, J. 80
Haddad, L. 80, 85
Hähnle, R. 248
Hall, D. 268
Hanyu, T. 30, 275
Hata, Y. 105, 288
Heider, R. 36
Hentschke, S. 174
Herrfeld, A. 174
Higuchi, T. 200
Hildebrand, L. 194
Hon-nami, A. 110
Hoshi, K. 200
Hozumi, T. 105
Hu, M. 118
Ike, T. 275
Ishizuka, O. 180
Janković, D. 18
Jaroszewicz, S. 24
Jóźwiak, L. 228
Jullien, G. A. 186
Kakusho, O. 105
Kalay, U. 268
Kameyama, M. 30, 275
Kimura, H. 30
Lang, C. 50
Lau, D. 85
Liu, R. 136
Lou, J. J. 66
Lukasiewicz, T. 236
Machida, H. 74
Maguire, L.P. 220, 294
Manyà, F. 248
McDaid, L.J. 294
McGinnity, T.M. 294
Milenović, D. 18
Miller, D. M. 214
Miyakawa, M. 256
Moraga, C. 36, 194
Mukaidono, M. 125, 214, 288
Nagata, Y. 214
Nakashima, K. 110
Ngom , A. 208
Ninomiya, T. 125
Olson, E. 42
Osman, M. 262
Perkowski, M. 50, 268
Pogosyan, G. 131
Rosenberg, I. G. 74
Saed, A. 186
Sasao, T. 59, 99, 166
Serfati, M. 10
Shen, J. 180
Simovici , D. 24
Sofronie-Stokkermans, V. 242
Stanković, R. S. 18, 154
Steinbach, B. 50
Stojmenović, I. 208
Takagi, N. 110
Tanno, K. 180
Thiele, H. 282
Webb, J. 220
Žunić, J. 208

Press Activities Board

Vice President and Chair:
Carl K. Chang
Dept. of EECS (M/C 154)
The University of Illinois at Chicago
851 South Morgan Street
Chicago, IL 60607
ckchang@eecs.uic.edu

Editor-in-Chief
Advances and Practices in Computer Science and Engineering Board
Pradip Srimani
Colorado State University, Dept. of Computer Science
601 South Hows Lane
Fort Collins, CO 80525
Phone: 970-491-7097 FAX: 970-491-2466
srimani@cs.colostate.edu

Board Members:
Mark J. Christensen
Deborah M. Cooper – Deborah M. Cooper Company
William W. Everett – SPRE Software Process and Reliability Engineering
Haruhisa Ichikawa – NTT Software Laboratories
Annie Kuntzmann-Combelles – Objectif Technologie
Chengwen Liu – DePaul University
Joseph E. Urban – Arizona State University

IEEE Computer Society Executive Staff
T. Michael Elliott, Executive Director and Chief Executive Officer
Matthew S. Loeb, Publisher

IEEE Computer Society Publications

The world-renowned IEEE Computer Society publishes, promotes, and distributes a wide variety of authoritative computer science and engineering texts. These books are available from most retail outlets. Visit the Online Catalog, *http://computer.org*, for a list of products.

IEEE Computer Society Proceedings

The IEEE Computer Society also produces and actively promotes the proceedings of more than 141 acclaimed international conferences each year in multimedia formats that include hard and softcover books, CD-ROMs, videos, and on-line publications.

For information on the IEEE Computer Society proceedings, send e-mail to cs.books@computer.org or write to Proceedings, IEEE Computer Society, P.O. Box 3014, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720-1314. Telephone +1 714-821-8380. FAX +1 714-761-1784.

Additional information regarding the Computer Society, conferences and proceedings, CD-ROMs, videos, and books can also be accessed from our web site at *http://computer.org/cspress*

1/29/99

Published by the IEEE Computer Society
10662 Los Vaqueros Circle
P.O. Box 3014
Los Alamitos, CA 90720-1314

IEEE Computer Society Order Number PR00161
IEEE Order Plan Catalog Number 99CB36329
ISSN 0195-623X
ISBN 0-7695-0161-3

ISBN 0-7695-0161-3