

AD-A218 008

DTIC FILE COPY

4

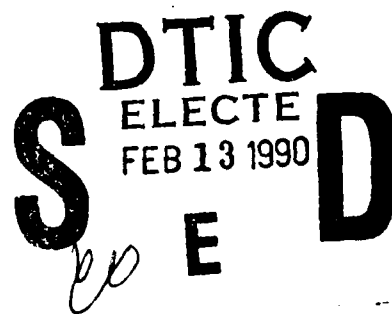
RADC-TR-89-259, Vol XI (of twelve)
Interim Report
October 1989



NORTHEAST ARTIFICIAL INTELLIGENCE CONSORTIUM ANNUAL REPORT - 1988 Inference Techniques for Knowledge Base Maintenance Using Logic Programming Methodologies

Syracuse University

Kenneth A. Bowen



APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

This effort was funded partially by the Laboratory Director's fund.

**ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700**

This report has been reviewed by the RADC Public Affairs Division (PA) and is releasable to the National Technical Information Services (NTIS) At NTIS it will be releasable to the general public, including foreign nations.

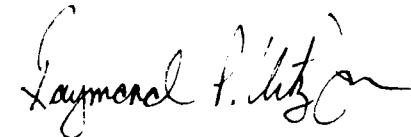
RADC-TR-89-259, Vol XI (of twelve) has been reviewed and is approved for publication.

APPROVED:



JOHH J. CROWTER
Project Engineer

APPROVED:



RAYMOND P. URTZ, JR.
Technical Director
Directorate of Command & Control

FOR THE COMMANDER:



IGOR G. PLONISCH
Directorate of Plans & Programs

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COES) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b. RESTRICTIVE MARKINGS N/A		
2a. SECURITY CLASSIFICATION AUTHORITY N/A			3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.		
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A					
4. PERFORMING ORGANIZATION REPORT NUMBER(S) N/A			5. MONITORING ORGANIZATION REPORT NUMBER(S) RADC-TR-89-259, Vol XI (of twelve)		
6a. NAME OF PERFORMING ORGANIZATION Northeast Artificial Intelligence Consortium (NAIC)		6b. OFFICE SYMBOL (if applicable)	7a. NAME OF MONITORING ORGANIZATION Rome Air Development Center (COES)		
6c. ADDRESS (City, State, and ZIP Code) Science & Technology Center, Rm 2-296 111 College Place, Syracuse University Syracuse NY 13244-4100			7b. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700		
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Rome Air Development Center		8b. OFFICE SYMBOL (if applicable) COES	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F30602-85-C-0008		
8c. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700			10. SOURCE OF FUNDING NUMBERS		
			PROGRAM ELEMENT NO. 62702F	PROJECT NO. 5581	TASK NO. 27
11. TITLE (Include Security Classification) NORTHEAST ARTIFICIAL INTELLIGENCE CONSORTIUM ANNUAL REPORT - 1988 Inference Techniques for Knowledge Base Maintenance Using Logic Programming Methodologies					
12. PERSONAL AUTHOR(S) Kenneth A. Bowen					
13a. TYPE OF REPORT Interim		13b. TIME COVERED FROM Jan 88 TO Dec 88		14. DATE OF REPORT (Year, Month, Day) October 1989	
15. PAGE COUNT 40					
16. SUPPLEMENTARY NOTATION This effort was funded partially by the Laboratory Directors' Fund.					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB-GROUP	Artificial Intelligence, Prolog High Level Language, Logic Programming Knowledge Base Maintenance, Truth Maintenance		
12	05				
19. ABSTRACT (Continue on reverse if necessary and identify by block number) The major focus of this year's effort has been on the development of a formal mathematical basis for logic programming. This report highlights four primary areas of on-going investigation into this research: 1) stratified knowledge bases, 2) the equivalence of non-classical logic programs, 3) multi-valued logic and logic programming, and 4) the topological aspects of logic programs. In addition, the MetaProlog (an extension of Prolog 1) and meta-logic programming are discussed.					
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED		
22a. NAME OF RESPONSIBLE INDIVIDUAL John J. Crowter			22b. TELEPHONE (Include Area Code) (315) 330-3564		22c. OFFICE SYMBOL RADC (COES)

DD Form 1473, JUN 86

Previous editions are obsolete.

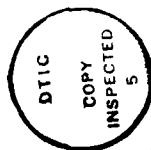
SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

UNCLASSIFIED

Item 10. SOURCE OF FUNDING NUMBERS (Continued)

Program Element Number	Project Number	Task Number	Work Unit Number
62702F	5581	27	23
61102F	2304	J5	01
61102F	2304	J5	15
33126F	2155	02	10
61101F	LDFP	27	01

Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	



UNCLASSIFIED

Northeast Artificial Intelligence Consortium

1988 Annual Report

Volume 11

Inference Techniques for Knowledge Base
Maintenance Using Logic Programming
Methodologies

Kenneth A. Bowen

Staff:

Hamid Bacha, Aida Batarek, Assoc. Prof. Howard Blair, Ilyas Cicekli,
V.S. Subrahmanian

Logic Programming Research Group
School of Computer and Information Science
313 Link Hall, Syracuse University
Syracuse, New York 13244-1240

This work was supported by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, New York 13441-5700, under Contract Number F30602-85-0008 which supports the Northeast Artificial Intelligence Consortium (NAIC).

Table of Contents

Executive Summary.....	2
11.1 Introduction	4
11.1.1 Non-Classical Logic Programming	6
11.1.2 Theoretical Basis for Logic Programming	6
11.1.3 MetaProlog	6
11.2 Hamid Bacha: Development of MetaProlog	8
11.3 Aida Batarek: Topological Aspects of Logic Programming	12
11.4 Howard Blair	14
11.5 Ilyas Cicekli: The Design and Implementation of the MetaProlog System	19
11.5.1 MetaProlog Theories	19
11.5.2 Abstract MetaProlog Engine	20
11.5.3 Proofs	21
11.5.4 Fail Branches	22
11.5.5 Garbage Collector	22
11.6 V.S. Subrahmanian	23
11.6.1 Logic Programming with Non-Classical Logics	23
11.6.2 Paraconsistent Reasoning	23
11.6.3 Topological Methods in Logic Programming	24
11.6.4 Metalogic Programming	24
11.6.5 Types in Prolog	24
11.6.6 Auto-Epistemic Logics	24
11.6.7 Nuclear Systems	24
11.6.8 Algebraic Theory of Logic Program Construction	25
11.6.9 Protected Completions of Logic Programs	25
11.6.10 Theorem Proving in Systems with Equality	25
11.7 Publications	27

Executive Summary

The primary focus of our research during the past year was on the development of a mathematical basis for logic programming. Specific areas of active research work include: stratified knowledge bases, equivalences of non-classical logic programs, multi-valued logic programming, and topological aspects of logic programming. Considerable success was achieved in each of these areas:

- We have studied the recursion theoretic complexity of the perfect (Herbrand) models of stratified logic programs. It is shown that these models lie arbitrarily high in the arithmetic hierarchy. As a consequence, we obtain a similar characterization of the recursion theoretic complexity of the set of consequences of a number of formalisms for non-monotonic reasoning. It is demonstrated that under certain circumstances, this complexity can be brought down to recursive enumerability.
- Logic programming with nonclassical logics has aroused a great deal of interest (cf. the proposals for incorporating quantitative "certainty factors" of Van Emden, and Subrahmanian, the three-valued logics of Lassez-Maher and Fitting, the four-valued logics of Blair-Subrahmanian and Fitting). We propose, four semantical notions of equivalences of sentences based on classical and non-classical logics. We show that under certain conditions on the lattice structure of the set of truth values of the logic of interest, three of these notions of equivalence can be captured in terms of results on the convergence of monotone nets in topology, while the fourth can be captured in terms of a property of convergent nets in compact Hausdorff spaces. We also show that these net convergence theorems allow us to characterize equivalences of Van Emden's quantitative rule sets, Lassez and Maher's three valued logic programs, and sentences with negation and disjunction as well as pure logic programs. Our work may be viewed as a semantical counterpart of Maher's syntactical characterization of pure 2-valued logic programs.
- We define a topology called the query topology on each of two sets – the set of interpretations of a first order language and the set of models of any sentence in the language. We show that in each of these cases, the resulting topology is a perfectly normal, T_4 -space. In addition, the query topology on the set of interpretations is compact. We derive a necessary and sufficient condition for the query topology on the space of models of sentences to be compact and show,

in addition, that the completions of canonical logic programs have a compact space of models. The familiar T_P operator may now be viewed as a function from a compact Hausdorff space to a compact Hausdorff space. We show that if P is either covered or function-free, then T_P is continuous in the query topology. The fact that the space of interpretations of a language is compact Hausdorff allows us to use the well-known theorems in topological fixed point theory to obtain heretofore unknown results on the semantics of logic programming. We present one such result – viz. a necessary and sufficient topological condition that guarantees the J -consistency (a notion defined in the paper) of completions of general logic programs.

- Recent results of Blair, Brown and Subrahmanian and independently, M. Fitting have shown that the declarative semantics of logic programs when interpreted over sets of truth values possessing some simple lattice theoretic properties shows remarkably little change. We prove here that the operational semantics (i.e. proof procedures) for such languages also show remarkably little change. The principal result is that under a natural condition of *support*, a straightforward generalization of SLD-resolution is sound and complete w.r.t. processing of queries over these differing logics.

11.1 Introduction

11.1.1 Non-Classical Logic Programming

Classical logic is a logic of truth. Using classical logic we can reason about the truth of different kinds of propositions relative to a given theory. Thus, we classify propositions as either being *false* or *true*.

Unfortunately, this often proves to be an overly simplistic point of view. For instance, the famous *Fermat's Last Conjecture* must assuredly be either true or false, but at this point in time, we are unable to say, with certainty, which of these cases is the correct one. The same is also true of the $P = NP$? problem. There are those, however, who strongly disbelieve the proposition $P = NP$. Classical logic does not permit us to express this *disbelief* because, of course, these disbelievers may well turn out to be wrong, and it may indeed be proven a few years hence that P is indeed equal to NP .

We view non-classical logics as logics of *belief*. Typically, human beings are fallible. They have beliefs and disbeliefs, which may be wrong when judged relative to some empirical standard. The process of changing our beliefs is a common occurrence in our daily world. Often our beliefs turn out to be correct, but often they are not. Worse still, people may hold beliefs that are inconsistent in some respects – yet they may be able to reason perfectly well about certain other domains.

Our point here is simply that the study of beliefs is important, and perhaps even more important than the study of truth, which is after all a rather ephemeral quantity. We will, in particular study:

- a theory of logic programming that allows us to reason in the presence of inconsistency. In particular we develop one such logic which belongs to a family of logics that go by the generic name *paraconsistent logics*.
- a theory of logic programming that allows us to reason in the presence of uncertain information, i.e. information which is vague in the sense that one is not sure of its truth/falsity, but has some feel (usually expressed in terms of a quantitative “certainty” factor) of the truth/falsity of a proposition.
- we have defined a family of programming languages over multivalued logics having a certain kind of algebraic structure (i.e. a complete lattice). We show

that under such circumstances, both the declarative (i.e. model theoretic and fixed point theoretic) semantics and the proof-theoretic (i.e. query processing procedures) generalize to the multivalued case.

11.1.2 Theoretical Basis for Logic Programming

As logic programming is a comparatively new field, we find that its basic theoretical underpinnings are very weak. There are many techniques in mainstream mathematics which may be used as *tools* to study the semantics of logic programming. It can hardly be doubted that establishing important links between well understood mathematical techniques and the semantics of classical and/or non-classical logic programming can only help enrich the semantics of logic programming. With this goal in mind, our group has undertaken the study of the topological and algebraic foundations of logic programming.

Associated with any (classical) logic program is an operator whose fixed-points are exactly the models of a formula called the completion of the program. One of the open problems in logic programming is to determine conditions for the completion to be consistent. A. Batarekh and V.S. Subrahmanian defined a (compact and Hausdorff) topology called the query topology on the space of interpretations of the first order language associated with a program. It is shown that whenever the program is either covered and/or function free, the operator associated with the program has a fixed-point iff it possesses a collapsibility condition. This collapsibility condition therefore yields a necessary and sufficient condition for program completions to be consistent (for such programs).

One can now study the algebraic properties of the space of programs by looking at the set of all operators associated with programs and associating some binary operators. Under some natural binary operators originally defined by Mancarella and Pedreschi, we obtain an algebra on programs that is easily seen to be a distributive lattice. The important question now is that of negation. Is there some notion of complementation relative to programs? Unfortunately, there is no such notion of complementation that yields a Boolean algebra or for that matter any richer algebraic structure like a ring, etc. (except in the most trivial cases). Finally, we can use this framework to study the equivalences of programs -- in particular, a notion called subsumption equivalence due to Maher can be generalized considerably to normal logic programs and also to paraconsistent and/or multivalued logic programs.

11.1.3 MetaProlog

The other major thrust of our group has involved the development of the MetaProlog system. MetaProlog is a powerful system whose primary aim is to allow the user

to fully utilise metalevel features like theory manipulation, proof maintenance, etc. Until recently, MetaProlog lacked an elegant theoretical basis, and this was one of the major open problems. During this year, V. S. Subrahmanian took the first steps towards the development of an elegant theoretical basis for MetaProlog. It was shown that the mathematical basis for metalogic programming is not very different from that for classical logic programming. However, certain essential changes are present.

11.2 Hamid Bacha

Development of MetaProlog

The two major accomplishments of this year are the completion of the first phase of the MetaProlog system and the implementation of a medical expert system in MetaProlog. The MetaProlog language is an extension of the popular logic programming language Prolog. As a high level programming language, Prolog has the most efficient implementation while still closely approximating the ideals of logic programming. Nevertheless, it has many limitations in terms of expressive power and problems with its ad hoc extra-logical features. These shortcomings have been recognized for a long time by many researchers, and a meta-level approach has been advocated as an alternative. Among the shortcomings that hamper the expressive power of Prolog are the many aspects that are supported by its underlying architecture, but not directly available to the user. Some of these aspects are:

- The sets of clauses (database)
- The provability relation ($\vdash$)
- The control strategy (depth first search, clause selection according to textual order)
- The rules of inference
- The proof trees

Some of the add hoc extra-logical features of Prolog that tend to cause problems are the "assert" and "retract" primitives which dynamically modify the database. The MetaProlog system tries to deal with some of these shortcomings while preserving the ideals of the logic programming paradigm. The tacit and otherwise inaccessible aspects of the system it makes explicit include the provability relation (referred to as "demo"), the sets of relations or procedures (referred to as "theories"), the sets of clauses making up a procedure (referred to as a "viewpoint"), and the proof trees. The primitives "assert" and "retract" are replaced by "addto" and "dropfrom" which are used to create new theories from existing ones. Some definitions were introduced to extend the accepted Prolog terminology to cope with the use of multiple databases (actually, instead of saying we use multiple databases, we prefer to say we have one database which contains multiple theories). These definitions are:

- A MetaProlog database is a collection of theories and relations (procedures).
- A relation is a collection of beliefs.
- A theory is a collection of viewpoints.
- A viewpoint is a set of related beliefs (equivalently, a subset of the set of beliefs making up a relation).
- A belief is a MetaProlog fact or rule.

As we can see from these definitions, the MetaProlog database contains theories, and the theories contain viewpoints. A built-in inheritance mechanism lets theories share clauses, thus avoiding the prohibitive cost of copying clauses from theory to theory. A fast algorithm is used to match the theories with their corresponding viewpoints.

Proofs, in the form of proof trees, are directly available to the MetaProlog user. They are treated as first-class objects and can be manipulated very much like any other MetaProlog terms. Since they include all the subgoals that participate in the evaluation of a given goal, they can be used, for example, to generate explanations for applications involving expert systems. An unexpected but pleasantly surprising use of proof trees is to affect the control strategy of the system by directing the search for a solution along a more desirable path. Indeed, if the system is presented with a goal and a proof tree indicating a possible solution, it only needs to check whether there is a proof for the stated goal along the branches of the search space corresponding to the given proof tree. In other words, the proof tree guides the search for the solution. No other possibly wrong or infinite paths need to be followed during the evaluation of the goal. No other solution needs to be considered. We can also use proof trees that are only partially instantiated. That is, a skeletal description of some desirable features we would like to see participate in the solution. In this case, the partially instantiated proof tree serves to focus the system's attention on specific portions of the search space, leaving it free to explore within these selected subspaces. Early pruning of non-fruitful branches of the search space and avoidance of blind alleys may lead to a more efficient solution for certain types of problems, despite the overhead associated with the proof trees.

The extensions mentioned above were achieved in the context of a compiled approach based on the Warren Abstract Machine architecture. This resulted in a fast and efficient system which relies on an interactive incremental compiler for flexibility

and ease of use. The objective of these extensions is to provide a richer and more expressive language, as well as a more accommodating environment for artificial intelligence applications such as knowledge representation, natural language processing, and expert systems.

To test the suitability of MetaProlog for large scale applications, we embarked on the task of implementing a medical expert system. The area of expertise selected was that of Acid-Base and Electrolyte Disorders. The goal was to integrate the clinical knowledge with the pathophysiological knowledge to come up with a robust expert system that combines both surface-level and deep-level reasoning. The system built used some innovative features such as:

- First-principles assisted evidential reasoning: This method relies on the more prevalent and widely used clinical knowledge for diagnostic purposes, but brings in the pathophysiological knowledge on an as needed basis.
- Progressively expanding diagnostic possibilities: Meta-level knowledge and priorities are used to restrict the search for the diagnosis to the more promising leads. These restrictions are then progressively lifted to include more and more possibilities for consideration. This method provides a more focussed approach and a better interaction between the system and the user.
- Thesaurus-driven user interface: all the interactions between the system and the user are carried through the user interface. To enhance the friendliness of the system, the user interface is coupled with a thesaurus that defines all the terms of interest in the domain of the expert system. The thesaurus specifies the type of query to be used with each term and the type of answer to expect. It lists the variations as well as the qualifiers applicable to each term. Whenever possible, it specifies the precondition that must hold before the user can be queried about a certain finding.

The preliminary results from our experiments with this system were very promising and seem to suggest that we have an adequate approach. More important, the whole experience in implementing this system point to the usefulness and suitability of MetaProlog for implementing expert systems. The MetaProlog system offers both a functional design advantage in terms of knowledge representation and hypotheses exploration, and a software engineering advantage in terms of structuring the expert system shell.

Future Work

Unlike many researchers who rely mainly on meta-interpreters to obtain the advantages of the meta-level approach, we went one step further and showed that it is possible to have some of these same advantages plus the speed of a compiler. However, only some of the desired features of MetaProlog have been implemented in this first phase. The next phase should address the following points:

- Explicit control: we should be able to specify the control regime to be used to solve any goal or subgoal. We should be able to choose between depth-first, incremental iterative deepening, or breadth first strategies. We should also have some way of specifying the order of the clauses when there are many alternatives.
- A choice of forward or backward chaining. This issue is tied to the control strategy above.
- A delay mechanism for waiting for some variables to be bound to ground terms.
- A mechanism for allowing coroutining to take place.
- Incomplete theories, that is theories that are not completely specified
- Explicit quantification

In addition to the direct work on the MetaProlog system, suitable projects in various areas of Artificial Intelligence should be identified and implemented in MetaProlog. These projects should be large and realistic enough to test the limits of the system. The lessons to be learned from these projects should hopefully confirm the viability of the many features of MetaProlog and help establish it as a major player in the area of research and development of Artificial Intelligence systems.

11.3 Aida Batarek

Topological Aspects of Logic Programming

Part I

Part I of this report was done in collaboration with V.S. Subrahmanian. A topology on the set of interpretations of a logic program P was defined, and its properties studied. The Query topology is defined as follows: the open sets are the collection of all subsets of X which satisfy a (possibly infinite) disjunction of individual existential queries. If all L_i 's are positive literals, the query is said to be positive, if all L_i 's are negative literals, the query is negative. We show that the Query topology gives rise to a totally disconnected, Tychonoff, complete and metrizable space.

A study of equivalences of sentences based on classical and non-classical logics was also pursued. We proposed four notions of equivalences of sentences. We showed that under certain conditions on the lattice structure of the set of truth values of the logic of interest, three of these notions can be captured in terms of results on the convergence of monotone nets in topology, while the fourth notion can be captured in terms of a property of convergent nets in compact Hausdorff spaces which is what the Query topology gives rise to. Our work may be viewed as a semantical counterpart of Maher's syntactical characterization of pure 2-valued logic programs. These results can be found in the technical report "Semantical Equivalences of (Non-Classical) Logic Programs", which has also been presented at the 5th International Conference on Logic Programming, August 88, Seattle.

Further investigations into the notion of axiomatizability, which we have previously defined, lead to the study of the special case of *finitely definite clause axiomatizability* or FDC-axiomatizability for short. We study mappings which are FDC-deformations, i.e., mappings from sets of interpretations into sets of interpretations such that the property of FDC-axiomatizability is preserved. We narrow the Query topology to the set of FDC-axiomatizable interpretations and investigate whether special properties can be obtained.

Part II

I studied the connection between the Query topology and the well-known Scott topology and established the following: the collection of all open sets which satisfy a disjunction of *positive* queries, are exactly the open sets in the Scott topology.

Similarly, the collection of all open sets which satisfy a disjunction of negative queries are the open sets in the Inverse Scott topology, which I have defined in a manner symmetrical to that of Scott topology. It is also shown that the Inverse Scott topology is distinct from the dual of the Scott topology. These results and some properties of the Query topology can be found in publication [9].

The lattice of interpretations is shown to be algebraic and supercontinuous hence also complete and continuous. Therefore one can compare the Query topology to the Lawson topology which is defined only on continuous lattices. The relationship between the two is established: the open sets in the Query topology are exactly the open sets in the Lawson topology. This in turn was used to prove that the space was compact and 0-dimensional, hence that it had a countable base of sets which are both open and closed. These results appear can be found in publication [10].

A notion of *axiomatizability* is introduced and a set Y of interpretations is shown to be axiomatizable with a set S of clauses if and only if Y is closed in the Query topology. These results and others pertaining to the applications of topology in Logic Programming have been collected in a technical report [11] which will be submitted to a journal for possible publication.

Having proved that the space of interpretations under the Query topology is a complete metric space, i.e. that there exists a metric which metrizes the space, the nature of the metric is investigated. I show that the Query topology is a Cantor space, and that there is a homeomorphism between the space of interpretations under the Query topology and the Cantor set.

The potential applications of the results found for the Query topology are studied with respect to non-monotonic deduction operators occurring in the underlying language. Specifically, an attempt is made at modifying the Query topology to deal with the non-monotonic operator ∇ introduced in my dissertation and used to introduce assumptions.

An investigation into the continuity properties (continuity as defined in topology) of the well known deduction operators $T \uparrow \alpha$ and $T \uparrow\uparrow \alpha$ shows that they are continuous over the set of Herbrand interpretations of a pure logic program P provided P has no clauses with free variables and $\alpha < \omega$.

11.4 Howard A. Blair

Several questions having to do with conservative extensions of logic programs were investigated during fiscal year 1988. These questions grew out of a paper by Howard Blair which was presented by him in August 1987 at the 1987 IEEE Symposium on Logic Programming, San Francisco, CA. The citation for this paper follows.

Blair, H. A. "Canonical Conservative Extensions of Logic Program Completions", *IEEE Symposium on Logic Programming*, San Francisco, August, 1987. pp. 154-161.

This work has culminated in a theorem which constructively establishes a domain over which every logic program is canonical. This theorem was found recently by Howard Blair in collaboration with Allen L. Brown, of both Xerox Webster Research Center's System Science Laboratory and the School of Computer and Information Science at Syracuse University.

Final editing of the following paper was carried out.

Apt, K. R., Blair, H. A., & Walker, A. "Towards a Theory of Declarative Knowledge," in *Foundations of Deductive Databases and Logic Programming*, Jack Minker, ed. Morgan-Kaufmann, Los Altos, CA. 1988. pp. 89-148.

An earlier version of this paper has issued in 1986 as an IBM Thomas J. Watson Research Center (Yorktown Heights) technical report. This was seminal work that introduced the theory of stratified logic programs. The book containing the paper was at last published, nine months behind schedule, in March 1988.

During the early part of the fiscal year an initial draft of the following paper completed.

Blair, H. A., Brown, A. L. and Subrahmanian, V. S. "A Logic Programming Semantics Scheme. Part I." Jan, 1988. Syracuse University Logic Programming Research Group Technical Report LPRG-TR88-8.

This paper presents the thesis that a logic program P without negation, over a variant logic, is a theory that can be associated with an operator whose prefixed points are exactly the models of P . Part II of this paper to be entitled, "A Logic Programming Semantics Scheme, Part II: DOXOLOG, a Belief Maintenance Language", is concerned with an application of the semantic approach of part I to give a formal semantics for the language DOXOLOG, indicated in the above title.

I continued my investigations of morphisms in logic programming model theory. This work is an attempt to, in particular, model-theoretically formalize the semantics of database updates. The idea is that a new database instance is a morphic (roughly homomorphic) image of a previous database instance such that both instances are models of the same theory of the database's integrity constraints.

A draft of a proposal for research on Computational Reasoning with Nonclassical and Paraconsistent Logics was prepared. The proposal is intended for submission to various funding agencies toward the end of the current fiscal year.

Working with Krzysztof R. Apt of the Centre for Mathematics and Computer Science in Amsterdam, the Netherlands, and the University of Texas at Austin, I wrote the following paper.

Apt, K. R. & Blair, H. A. "Arithmetic Classification of Perfect Models of Stratified Programs" Jan. 1988. Appears in *The Proceedings of the Joint Fifth International Logic Programming Conference and Fifth IEEE Symposium on Logic Programming* Seattle, Washington, August, 1988. Also appears as the Syracuse University Logic Programming Research Group Technical Report LPRG-TR88-11.

This paper shows that the expressive power of stratified programs climbs the arithmetic hierarchy as the number of strata increase. This and related results were discussed but not proved in earlier drafts of our paper "Towards a Theory of Declarative Knowledge", mentioned above. A subsequent result showed that if the programs satisfied an additional hierarchical constraint, then their standard model remains computable, hence queries are computable in such circumstances. These latter results were reported in :

Apt, K. R. & Blair, H. A. "Recursion-free Programs". Syracuse University Logic Programming Research Group Technical Report LPRG-TR-88-12.

Following the Logic Programming conference last August, the "Arithmetic Classification" paper was invited for submission to the journal *Fundamenta Informatica*, and a version that amalgamates the conference paper and the "Recursion-free Programs" paper was subsequently accepted for a special issue on stratified logic programs and databases. This work was also presented by me at a colloquium talk in April 1988 at the State University of New York at Albany.

My student V.S. Subrahmanian travelled to Pune, India to present the paper indicated below.

Blair, H. A. & Subrahmanian, V. S. "Paraconsistent Logic Programming" (Preliminary Version) Seventh Conference on Foundations of Software Technology & Theoretical Computer Science. December, 1987. pp. 340-360.

Subsequently, in February, 1988 a revised version of this paper was invited for submission to a special issue of the journal *Theoretical Computer Science* for a special issue on selected papers from the Seventh Conference on Foundations of Software Technology & Theoretical Computer Science. The paper was subsequently accepted. Paraconsistency is discussed somewhat further by V.S. Subrahmanian, below.

Progress on two lines of research was made which were subsequently embodied in revisions of the following two papers.

Blair, H. A., Brown, A. L. and Subrahmanian, V. S. "A Logic Programming Semantics Scheme, Part I." Jan, 1988. Syracuse University Logic Programming Research Group Technical Report LPRG-TR88-8.

Blair, H. A. "Metalogic Programming and Direct Universal Computability". Syracuse University Logic Programming Research Group Technical Report LPRG-TR88-23.

This paper was presented at *Meta88: Workshop on Meta-programming in Logic Pro-*

gramming. The paper is scheduled to appear in the proceedings of this workshop to be published by MIT Press.

My second Ph.D. student, Toshihiro Wakayama passed his qualifying examination, both the written and oral parts, and formally became a doctoral candidate. Toshihiro is currently in his third year as a Graduate Fellow. While not directly supported by the grant, Toshihiro has been an integral part of the Logic Programming Laboratory's effort. Toshihiro also had the following paper accepted for the 1988 Conference on Automated Deduction. He is expected to graduate at the end of the current academic year.

Payne, T. H. & Wakayama, T. "Case Inference in Resolution-Based Languages," *Proc. 9th Conference on Automated Deduction, Lecture Notes in Computer Science*, Springer-Verlag, May 1988.

Toshihiro subsequently presented the paper at the conference.

My third Ph.D. student, Frank Yang took and passed his Ph.D. qualifying exam. Frank has begun research on relevance and relatedness logic, and will be supported in the Logic Programming Laboratory through indirect assistance of a major corporation active in computer science research. As Frank's work progresses it is expected that he will come more and more under the tutelage of Allen Brown.

My fourth Ph.D. student, Marion Ben Jacob, passed both the written and oral parts of her Ph.D. qualifying examination thereby formally becoming a doctoral candidate. Marion, in collaboration with Prof. Melvin Fitting, CUNY, jointly authored a paper entitled "Stratified and Three Valued Logic Programming Semantics". This paper also appeared in the proceedings of the Joint Fifth International Logic Programming Conference and Fifth IEEE Symposium on Logic Programming, and will appear in the special issue of *Fundamenta Informatica* on stratified Logic Programs and databases.

Some progress was made early in 1988 on "An Inductive, Stratification-free Definition of Standard Models of Stratified Logic Programs". It is still in a formative stage. The work that needs to be done to establish this 'definition' requires that a limit of

an alternating operator, recursive in zero-jump, exist in the right circumstances.

In collaboration with Prof. Mark Brown of the Philosophy Dept., Syracuse University, I organized an interdepartmental weekly seminar on mathematical logic. My general interest in a theory of intentionality involves considering formal metalogics, which in turn, due initially to the work of Peter Aczel of Manchester University and Jon Barwise and his colleagues at the Center for the Study of Language and Information at Stanford University, involves what has come to be known as situation theory and non-well-founded set theory. The mathematical logic seminar has been studying carefully some of the work of Barwise and his colleagues. My own research yielded techniques for constructively obtaining what are called *co-inductively defined* classes of hypersets, which include well-founded and non-well-founded sets, and in one case with the aid of my techniques we have shown that the class of hyperordinal numbers contains all hypersets, a new, though simple, result, and the same techniques indicate how to modify the definition of *hyperordinal* to obtain a less trivial class which will still include all of the usual, well founded ordinals. I am currently at work on this modification and constructing the theory to justify it. I am expecting that the results obtained here will lead to theories of metalogic and intentionality which in turn I can exploit for the purpose of formulating a continuous semantics for logic programming languages and metalogic programming languages.

Two other results obtained during Fiscal year 1988 were that I developed a functionally oriented theory of nondeterministic partial recursive functions in accord with an earlier theory, relationally oriented, of such functions advanced by Ashok Chandra, and this theory was applied to showing that a logic program with a well-founded dependency relation forms a Π_1^1 -complete set.

11.5 Ilyas Cicekli

The Design and Implementation of The MetaProlog System

Most of the meta-level systems implemented in last decade are meta-level interpreters which introduce extra interpretation layers that slow down the execution. The MetaProlog system described in this report is a compiler-based meta-level system for the MetaProlog programming language. Since MetaProlog is an extension of Prolog, we extended the Warren Abstract Machine (WAM) to the Abstract MetaProlog Engine (AMPE). MetaProlog programs are directly compiled into the instructions of the AMPE.

In the rest of this report, the MetaProlog system is briefly described. Theories which are first class objects in MetaProlog, and their representations in the MetaProlog system are discussed in Section 2. The basic structure of the AMPE is explained in Section 3. In the last section, the garbage collector of the MetaProlog system is presented.

11.5.1 MetaProlog Theories

In Prolog, there is a single database, and all goals are proven with respect to this database. When there is a need to update this database, the builtins `assert/retract`, which are ad hoc extensions to the basic logic programming paradigm, are used to create the new version of this database by destroying the old database in the favor of the new one. On the other hand, there can be more than one theory in MetaProlog, and a goal can be proven with respect to one of these theories. A new theory in MetaProlog is created from an old theory without destroying the old theory.

A new theory is created from an old theory already exists in the system by adding some clauses or dropping them. The new theory inherits all procedures of the old theory except procedures explicitly modified during its creation. Although we create a new theory from an old theory, the old theory is still accessible by the user.

The provability relation between a theory and a goal is explicitly represented in MetaProlog by a two argument predicate `"demo"`. The relation `"demo(Theory,Goal)"` precisely holds when `"Goal"` is provable in `"Theory"`. Similarly, the relation `demo(Theory,Goal,Proof)` holds when `"Proof"` is the proof of `"Goal"` in `"Theory"`. When one of these provability relations is encountered, the underlying theorem prover tries

to prove the given goal with respect to the given theory.

Theories of the MetaProlog system are organized in a tree whose root is a distinguished theory, the base theory. The base theory contains all the system builtins, and all other theories in the system are descendants of the base theory. In other words, all theories can access procedures of the base theory.

Every theory in the MetaProlog system possesses a default theory except for the base theory. The default theory of a theory T is the theory where we search for a procedure if the search for that procedure in T fails. This search through default theories continues until the procedure is found or the base theory is reached.

To shorten the depth of the theory, theories in the MetaProlog system are classified into two groups : "default theories", and "non-default theories". A "non-default theory" is a theory that carries information about all procedures that underwent modifications in the ancestor theories between this theory and its default theory. Access to these procedures is very fast, at expense of copying some references. The default theory of a theory is the first ancestor theory that is a "default theory". A "default theory" is a theory whose descendants don't carry any information about the procedures occurring in that theory. If only default theories are used, access to a given procedure in a given theory may require a search through all its ancestor theories. In this case, access to a procedure may be slow, but no copying of references is needed. Depending on the problem, the system tries to use one or the other approach, or a combination of both to achieve a balance between speed of access and space overhead.

When a new theory is created from a non-default theory, its default theory will be its father's default theory. But if a new theory is created from a default theory, its default theory will be its father. In the first case, the new theory will be at its father's level. In the second case, the new theory will be at one level above its father's level. Thus we don't increment the depth of the theory tree when a theory is created from a non-default theory.

11.5.2 Abstract MetaProlog Engine

Our main goal in this project was to create an efficient compiler-based MetaProlog system. Since MetaProlog is an extension of Prolog, the Warren Abstract Machine (WAM) was the best starting point. For this purpose, the WAM is extended to the Abstract MetaProlog Engine (AMPE).

The AMPE performs most of the functions of the WAM, but it also has some extra features to handle theories and compiled procedures as data objects of the system.

These extra features basically are:

- Extra registers to handle theories in MetaProlog.
- A different memory organization which is more suitable to handle compiled procedures and theories as data objects of the system.
- The functions of the procedural instructions in the AMPE differ from their functions in the WAM.

There are two new registers in the AMPE in addition to the registers the WAM does. The first one is the "theory register" which holds the current theory (context) of the MetaProlog system. The value of the "theory register" is changed when the context of the system is switched to the another context. This register is also saved in choice points so that the context of the system can be restored the value saved in the last choice point during backtracking. The second one is the "theory counter register" which is simply a counter to produce a unique theory-id for each theory in the system. It is incremented to indicate the next available theory-id after the creation of each theory.

The code space and the heap in the WAM are integrated as a single data area in the AMPE which is more suitable to handle compiled procedures as data objects. This integrated space in the AMPE is still called "heap". Thus theories and compiled procedures can be created on fly, and they are can be easily discarded when the need for them is gone. The local stack and the trail of the AMPE still perform the same job they perform in the WAM.

11.5.3 Proofs

The AMPE can run in two different modes. When a two argument "demo" predicate is encountered, the system runs in the simple mode. In the simple mode, the system only proves a goal with respect to the current theory of the system. When a three argument "demo" predicate is encountered, the mode of the system is switched to the proof mode. In the proof mode, a goal is not only proved with respect to the current theory of the system, its proof is also collected. At the implementation level, the mode of the system is represented by a mode flag which is also saved in choice points so that the system can switch from one mode to the another during backtracking.

In the simple mode of the system, only the core part of the system described above is used. On the other hand, two extra registers are used in addition to the core part of the system when the system runs in the proof mode. These extra two registers are used to collect the proof of a goal during its execution.

11.5.4 Fail Branches

After finishing the core part of the MetaProlog system, I started to extend the MetaProlog system which can handle extra control information in the demo predicate. Now, the MetaProlog system have the following capabilities.

1. Now the system can get fail branches of a goal in addition to its success branches (proofs). When the goal "demo(T,G,branch(P))" is submitted, P is unified with a branch (fail or success) of the proof tree of G in T. On the other hand, when the goal "demo(T,G,proof(P))" is submitted, P is unified with only a success branch of the proof tree of G in T.
2. The system also supports a fourth argument demo whose fourth argument is control information. In some cases, to get a complete proof of a goal can be unnecessary. We may not need all proofs of subgoals. For this purpose, proofs of these subgoals can be skipped by using the following form of the demo predicate.

demo(T, G, proof(P), skip_proofs_of(Subgoals))

After the execution of the above, proofs of SubGoals don't appear in the proof P of G in T.

11.5.5. Garbage Collector

The garbage collector of the MetaProlog system collects all the garbage in the system including the garbage in the code. It consists of a recursive marking routine and a compaction routine. The marking routine recursively marks all locations in the heap which are accessible from external locations such as argument registers, and locations in the local stack. The garbage compaction routine, an extension of Morris's compaction algorithm, adjusts all pointers in the uncompact heap and does the real compaction.

11.6 V. S. Subrahmanian

Theory of Logic Programming

My primary work during this period concentrated on the development of a mathematical basis for classical and non-classical logic programming. In particular, I developed, jointly with Aida Batarek, a topological theory of logic programming model theory, while both alone and/or jointly with A. N. Hirani, I developed an algebraic basis for logic programming. I also concentrated on the study of several different non-classical logic programming languages.

11.6.1. Logic Programming with Non-Classical Logics.

I have been involved in the development of a family of non-classical logic programming languages that can be semantically characterized in terms of fixed-point theory. Proposals for logic programming with specific logics (e.g. quantitative logics, paraconsistent logics, etc.) were later generalized to yield a generalized declarative semantics for logic programming over certain kinds of partially ordered sets of truth values. This declarative semantics is independent (to some extent) of the syntactic nature of a non-classical logic program. In addition, I developed a proof-theoretic generalization of SLD-resolution that is sound and complete for many-valued logic programs (whose set of truth values is a complete lattice).

11.6.2. Paraconsistent Reasoning.

The design of very large knowledge bases may sometimes result in some inaccuracies. Paraconsistent logics provide a framework for reasoning in the presence of inconsistency (in the sense of classical logic) via non-classical model theory. Howard Blair and I have worked on a formal theoretical framework for mechanical reasoning in the presence of inconsistency. More recently, M. Chakrabarti and I are working on the semantics of general logic programs (even those whose completions are inconsistent) with a view to developing a theory of local and global consistency. Newton da Costa and I are investigating syntactic consequence relations that lead to paraconsistent logics with a view to developing a proof-theoretic characterization of inconsistent databases.

11.6.3. Topological Methods in Logic Programming.

Aida Batarekh and I studied the topological properties of the space of models of logic programs (and also arbitrary sentences in first order logic). We then derived results on the fixed-points of non-monotonic operators that map structures to structures. As a consequence of some results on the (topological) continuity of the well-known operator T_P associated with a logic program P , we were able to obtain necessary and sufficient conditions on the consistency of $comp(P)$ (when P is either a function free or covered logic program).

11.6.4. Metalogic Programming.

My paper *Foundations of Metalogic Programming* is the first paper to address the problem of developing a formal theoretical framework for reasoning about the amalgamation of object language and metalanguage in logic programming. It is a companion to the paper by Pat Hill and John Lloyd that considers metalevel programming *without* the amalgamation.

11.6.5. Types in Prolog.

Lee Naish and I have jointly developed a framework for incorporating types in Prolog. For programming purposes, our view is that type declarations are useful, and our semantics essentially characterizes logic programming augmented with type declarations.

11.6.6. Auto-Epistemic Logics.

Wiktor Marek and I are currently studying the connections between differing treatments of negation in logic programming and AI. In addition, we have studied the complexity of determining the truth of a formula in a stable expansion of an auto-epistemic first order theory.

11.6.7. Nuclear Systems.

A nuclear system is essentially a triple $\langle S, \vdash, Q \rangle$ where S is a non-empty set, Q is the set of existential queries that can be expressed in some fixed but arbitrary first order language, and $\vdash$ is a binary relation between S and Q . For example, S may be a set of theories, and $\vdash$ may be an entailment relation, or S may be a set of interpretations for a first order language and $\vdash$ may be a model-theoretic satisfaction relation, or S may be a set of theories and $\vdash$ may be a non-monotonic forcing relation. When the nuclear system satisfies some simple conditions, S turns out to be a compact Hausdorff space (under a topology induced by the $\vdash$ relation). One can now study the fixed-points of non-monotonic closure operators in terms of topological results.

11.6.8. Algebraic Theory of Logic Program Construction.

Given a logic program P , the operator T_P associated with P is closely related to the intended meaning of P . Given a first order language L that is generated by finitely many non-logical symbols, our aim is to study the algebraic properties of the set $\{T_P \mid P \text{ is a general logic program in language } L\}$ with certain operators on it. For the operators defined in this paper the resulting algebraic structure is a bounded distributive lattice. Our study extends (to the case of general logic programs), the work of Mancarella and Pedreschi who initiated a study of the algebraic properties of the space of pure logic programs. We study the algebraic properties of this set and identify the ideals and zero divisors. In addition, we prove that our algebra satisfies various *non-extensibility* conditions. This algebraic study shows promise of leading to a theory of modules in logic programming.

11.6.9. Protected Completions of Logic Programs.

The notion of protected completion $pc(P)$ of a logic program P was introduced by Jack Minker and Don Perlis. The Minker-Perlis proposal laid the foundation for reasoning via protected completions for pure, function free logic programs. We extend their work by characterizing protected completions of general logic programs. Thus, both restrictions in the Minker Perlis proposal are removed. Operational algorithms are also developed. This work is being carried on jointly with James Lu.

11.6.10. Theorem Proving in Systems with Equality.

James Lu and I studied certain open problems concerning the soundness and completeness of various problems in RUE-NRF deduction. We proved, amongst other results, that RUE-NRF deduction in strong form is incomplete contradicting existing published results of V. Digricoli and M. Harrison. Our disproof has since been acknowledged as being correct by V. Digricoli. Since then, we have worked on the problem of termination of the viability check in RUE-NRF deduction using a method based on AND/OR graphs.

11.7 Publications

1. Apt, K. R., Blair, H. A., & Walker, A. "Towards a Theory of Declarative Knowledge," in *Foundations of Deductive Databases and Logic Programming*, Jack Minker, ed. Morgan-Kaufmann, Los Altos, CA. 1988. pp. 89-148.
2. Apt, K. R. & Blair, H. A. "Arithmetic Classification of Perfect Models of Stratified Programs" Jan. 1988. Appears in *The Proceedings of the Joint Fifth International Logic Programming Conference and Fifth IEEE Symposium on Logic Programming* Seattle, Washington, August, 1988. Also appears as the Syracuse University Logic Programming Research Group Technical Report LPRG-TR88-11.
3. Apt, K. R. & Blair, H. A. "Recursion-free Programs". Syracuse University Logic Programming Research Group Technical Report LPRG-TR-88-12.
4. H. Bacha. *MetaProlog Design and Implementation*, Proceedings of the Fifth International Conference on Logic Programming. Seattle, Wa. 1988. Edited by K.A. Bowen and R. Kowalski.
5. H. Bacha. *Beyond the WAM: A PAM for the CAM. (A Prolog Abstract Machine for Content-Addressable Memory.)*, Submitted to the 6th International Conference on Logic Programming to be held in Lisbon, Portugal in June 1989.
6. H. Bacha and S. Khanna. *Program Verification Using Meta-Level Logic Programming*, Submitted to the 6th International Conference on Logic Programming to be held in Lisbon, Portugal in June 1989.
7. H. Bacha, K. Bowen and C. Carvounis. *Clinical vs Pathophysiological Knowledge in Medical Expert Systems*, In preparation, To be submitted to a medical journal.
8. A. Batarekh and V. S. Subrahmanian. *Semantical Equivalences of (Non-Classical) Logic Programs*, Proc. 5th International Conference on Logic Programming, Seattle, MIT Press.
9. A. Batarekh and V.S. Subrahmanian. *The Query Topology in Logic Programming*, Proc. Intl. Symp. on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science, Springer Verlag, Feb. 1989.
10. A. Batarekh and V.S. Subrahmanian. *Topological Model Set Deformations in Logic Programming*, to appear in: *Fundamenta Informatica*.

11. A. Batarek and V.S. Subrahmanian. *A T_4 Space of Models of Logic Programs and their completions, I: Foundations*, Technical Report, Logic Programming Research Group, LPRG-TR-88-15.
12. Blair, H. A. "Canonical Conservative Extensions of Logic Program Completions", *IEEE Symposium on Logic Programming*, San Francisco, August, 1987. pp. 154-161.
13. Blair, H. A. "Metalogic Programming and Direct Universal Computability". Syracuse University Logic Programming Research Group Technical Report LPRG-TR88-23. To appear in the *Proceedings of the Workshop on Metalogic Programming 1988* MIT Press.
14. Blair, H. A., Brown, A. L. and Subrahmanian, V. S. "A Logic Programming Semantics Scheme, Part I." Jan, 1988. Syracuse University Logic Programming Research Group Technical Report LPRG-TR88-8.
15. H. A. Blair and V.S. Subrahmanian. *Paraconsistent Logic Programming*, Proc. 7th Conference on Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Computer Science, Vol. 287, pps 340-360, Springer Verlag. Extended version to appear in: *Theoretical Computer Science*.
16. H. A. Blair and V. S. Subrahmanian. *Strong Completeness Results for Paraconsistent Logic Programming*, submitted to a technical journal.
17. H. A. Blair and V. S. Subrahmanian. *Paraconsistent Foundations for Logic Programming*, to appear in: *Journal of Non-Classical Logic*.
18. I. Cicekli. *An Abstract MetaProlog Engine for MetaProlog*, in: Proc. of the Workshop on Meta-Programming in Logic Programming, Bristol, England, June 1988.
19. A. N. Hirani and V. S. Subrahmanian. *Algebraic Foundations for Logic Programming, I: The Distributive Lattice of Logic Programs*, to appear in: *Fundamenta Informatica*.
20. Payne, T. H. & Wakayama, T. "Case Inference in Resolution-Based Languages," *Proc. 9th Conference on Automated Deduction, Lecture Notes in Computer Science*, Springer-Verlag, May 1988.

21. V. S. Subrahmanian. *Query Processing in Quantitative Logic Programming*, Proc. 9th International Conference on Automated Deduction, (eds. E. Lusk and R.Overbeek), Lecture Notes in Computer Science Vol. 310, pps 81-100, Springer Verlag.
22. V. S. Subrahmanian. *Intuitive Semantics for Quantitative Rule Sets*, Proc. 5th International Conference/Symposium on Logic Programming, (eds. K.Bowen and R.Kowalski), MIT Press, Aug. 1988.
23. V. S. Subrahmanian. *Mechanical Proof Procedures for Many Valued Lattice Based Logic Programming*, accepted for publication in a technical journal.