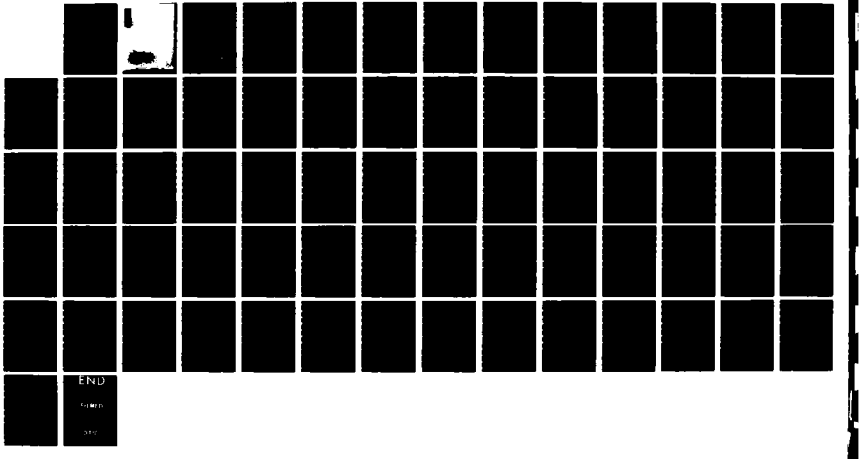
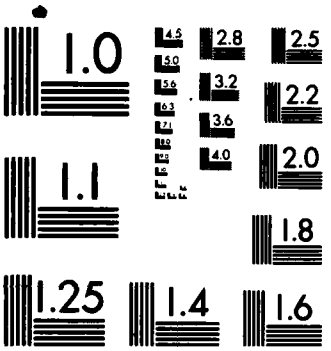
AD-A161 249 RADICAL COMPUTING III(U) MITRE CORP MCLEAN VA 1/1
S AMAREL ET AL. SEP 85 JSR-84-701 F19628-84-C-0001

UNCLASSIFIED F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

JASON

Radical Computing III

Saul Amarel
Curtis G. Callan, Jr.
Roger F. Dashen
Alvin M. Despain
Oscar S. Rothaus

September 1985

JSR-84-701

JASON
The MITRE Corporation
1820 Dolley Madison Boulevard
McLean, VA 22102

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered) UNCLASSIFIED

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: JSR-84-701	2. GOVT ACCESSION NO.: AD-A161249
3. RECIPIENT'S CATALOG NUMBER:	
4. TITLE (and Subtitle): RADICAL COMPUTING III	5. TYPE OF REPORT & PERIOD COVERED: Technical
	6. PERFORMING ORG. REPORT NUMBER:
7. AUTHOR(s): Saul Amarel, Curtis G. Callan, Jr., Roger F. Dashen, Alvin M. Despain, Oscar S. Rothaus	8. CONTRACT OR GRANT NUMBER(s): F19628-84-C-0001
9. PERFORMING ORGANIZATION NAME AND ADDRESS: The MITRE Corporation, 1820 Dolley Madison Blvd., McLean, VA 22102	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 8503Z
11. CONTROLLING OFFICE NAME AND ADDRESS: DARPA, 1400 Wilson Boulevard, Arlington, VA 22209	12. REPORT DATE: September 1985; 13. NO. OF PAGES: 65
14. MONITORING AGENCY NAME & ADDRESS (if diff. from Controlling Office):	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT (of this report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Molecular computing, advanced computing techniques, computing

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

In this report, we explore two topics suggested from the last study. The first is Molecular Computing which may offer a technology that could realize some of the ideas of reversible computing. The second topic is the Theory of Transformation of Weak Methods into Strong Methods. This is a direct extension of our prior Source Program Transformation work.

Since the goal of this and all our previous work is to identify potential new development, the ideas discussed herein are quite speculative and should be considered with some caution.

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

19. KEY WORDS (Continued)

20 ABSTRACT (Continued)

We limit our investigation to improvements in computer performance in the execution of only difficult calculations. We will not seek higher performance for problems easily managed by today's machines.

ACKNOWLEDGEMENT

The field of molecular computing has been developed, nurtured and supported by Forest Carter of NRL. Dr. Carter worked with us on this study and provided research materials as well as his individual insights into molecular computing. We are very grateful for his help in this study. We are also grateful for discussions with Pepi Ross of SRI and James Clark Solinsky of IRI. We also would like to acknowledge our colleagues in JASON for their contribution of inspiring discussions and helpful criticism. In particular, the advice of Ken Case, Doug Eardley, Allen Peterson, and John Vesecky is especially appreciated.

TABLE OF CONTENTS

1. INTRODUCTION

1.1. History

This is a report of the third JASON summer study of advanced computing techniques that could lead to a radical improvement in computer performance. The purpose of these studies is to identify trends, indicators, and potential techniques that might someday lead to such performance. In the first study we investigated *Residue Arithmetic* and *Symbolic Computing[1].* In the second year we extended our study of symbolic computing into *Source Program Transformation* and also investigated a new topic, *Reversible Computing[2].*

1.2. Scope

In this report, we explore two topics suggested from the last study. The first is *Molecular Computing* which may offer a technology that could realize some of the ideas of *reversible computing.* The second topic is the *Theory of Transformation of Weak Methods into Strong Methods.* This is a direct extension of our prior *Source Program Transformation* work.

Since the goal of this and all our previous work is to identify potential new developments, the ideas discussed herein are quite speculative and should be considered with some caution.

We limit our investigation to improvements in computer performance in the execution of only *difficult* calculations. We will not seek higher performance for problems easily managed by today's machines.

1.3. Difficult Problems

In our view, difficult problem domains are characterized by these factors:

(1) Massive numerical calculations on large data structures.

(2) Search for an optimal (or near optimal) solution over a problem space.

(3) Symbolic calculations, tightly coupled to the numeric calculations and the search process.

Problems of this type demand a computer architecture with massive main memory, fast digital circuits and special structure for supporting symbolic and search calculations.

1.4. Limitations to High Performance

The rate at which computers can execute is limited by the serial nature of today's computer programs, the strength (or efficiency) of the algorithms employed, and the physical size of the components.

The joint problem of overcoming the serial nature of todays computing tasks and the expressing, scheduling and executing parallel calculations is one of the great unsolved problems of computer science. While it is very important, we have chosen not to address it in this particular report. We note however, that the language Prolog, employed ir this report, has great potential for parallel execution[3].

We will discuss the theory of strength improvement in some detail. This is but one aspect of the efficiency problem, however it is the most important for the problem domain we have in mind.

The primary limitation to circuit speed is the finite speed of light (Maxwell's equations) and the need to dissipate heat. Smaller components generate less heat and therefore can be packed closer together, decreasing the time it takes to propagate signals between themselves. Smaller computers can, all other factors being equal, run proportionally faster than larger ones. Today, the basic component of computers, the logical switch, is the transistor. If this switch could be reduced in size from about 10^{-5} meters to a molecular scale of about 10^{-8} meters, a speed-up of perhaps as much as 1000 times might be achievable. The possibility of this potential speed-up motivated this study of molecular computing.

In modern electronic devices, response time scales linearly with size (See p. 33 of [4]). However the main argument for linear speedup is the simple argument that to first-order, transit time, across a device, is proportional to device size, whatever the device.

2. MOLECULAR COMPUTING

2.1. Introduction

Feynman[5] may have been the first to seriously consider the prospects of building computers on a molecular scale. He observed:

> ....if computers had millions of times as many computing elements as they do now, they could make such judgements (as in image recognition). But with the present size of components, these computers would fill millions of rooms, cost impossible sums of money, and be too big to work anyhow. (In the 'smaller' computers) ... the wires should be ten or 100 atoms in diameter, and the circuits should be a few thousand Angstroms across.

Thus, the basic idea of molecular computing is to radically reduce (i.e. to molecular scales) the size of the components that make up today's computers. This means that the finest features, today about 10^{-6} meters would be reduced to about 10^{-9} meters. Thus switches, which today are about 10^{-5} meters across, might be reduced to the range of 10^{-7} to 10^{-8} meters. If such molecular components could be developed, then perhaps up to 10^{18} bits/cm^3 of storage and down to 10^{-12} seconds/gate of delay time might be achieved. Today, VLSI circuits achieve about 10^{6} bits/cm^3 and about 10^{-9} seconds/gate. It is, of course, the *potential* for this dramatic improvement in storage density and circuit performance that motivates this study. The reader should be cautioned however, that the ideas on molecular computing are speculative. There is very little experimental or theoretical work to support most of the interesting ideas.

The modern ideas of Molecular Electronic Devices (MED) are the primary source of ideas for molecular computers. These have been nurtured by Forrest L. Carter of the Navel Research Laboratory since 1979. He has written extensively on the ideas[6, 7, 8, 9, 10, 11, 7, 12] He has also sponsored several workshops on Molecular Electronic Devices.[13, 7]. There is a wealth of material available from these workshops and in the papers referenced by the workshop papers. The reader is encouraged to examine these materials, although we will mention topics covered in these materials only so far as they apply to the issues we discuss here.

2.2. Approaches

There are a number of approaches to molecular computing:

(1) Soliton propagation

(2) Conduction electrons

(3) Super conduction

(4) "Neuron-like", biological based elements

(5) Chemical reactions, similar to DNA etc. reactions.

We will discuss the first three possibilities. We will *not consider the chemical or direct biological approaches,* such as nerve conduction, because they are too slow [1] It is, however, possible to employ biological assembly techniques for "fast" non-biological circuits.

2.3. Construction Techniques

Feynman[5, 14] also discussed possible construction techniques in his pioneering papers. In particular he said[5]

> One possibility we might consider, since we have talked about writing by putting atoms down in a certain arrangement, would be to evaporate metals or other materials in successive layers until we have a block of exceedingly fine dimensions.
> ... The electrical equipment won't simply be scaled down; it has to be redesigned. But I can see no reason why it can't be redesigned to work again.

We concur. There seem to be several possible avenues for molecular construction:

(1) Lithography

(2) Modular Chemistry

(3) Monolayer Films

---

[1] The response time of nerve cells ($\sim 10^{-2}$ sec) is very slow compared to that of modern electronic circuits ($\sim 10^{-9}$ seconds). Of course, if the fast circuits are not realizable, then the chemical and nerve conduction approaches should be investigated. From where technology now stands, it appears that the fast circuit techniques are no more difficult than any of the alternatives, and therefore become the choice for current investigation.

(4) 'Self-Assembly' (Biological Techniques)

The reader is referred to the workshop papers mentioned above for details of all these techniques. There are undoubtedly other possible approaches that might be applied if these techniques will not suffice. The important point is that there seem to be a number of possible techniques that could possibly be developed to assemble molecular computers.

2.4. Machine Organization

2.4.1. Fault Tolerance

The machine organization of molecular computers is likely to be very different from today's computers due to the very high failure rates that must be expected for molecular components. Cosmic rays for example, unstoppable with any practical shielding, will regularly destroy portions of the molecular circuits. Manufacturing techniques are not now, nor are they ever, likely to be perfect, and repairs will perhaps be impossible. Thus a great redundancy of molecular circuits must be provided. Error (i.e. failure) detection and correction circuits will be needed at all levels of molecular circuits and components. This is a reasonably well understood technical problem and does not pose any unsurmountable difficulties. Since this is not a critical issue, we will not discuss error and failure problems any further in this report.

2.4.2. Finite State Automata

In other respects, the organizational principles of today's computers should be applicable to molecular computers. An exception might occur if, for some currently unseen reason, some molecules are found that are very efficient finite state machines that can be organized into an array of cellular automata. Array cellular automata, at least as they are understood today, are inefficient organizations for doing calculations and would not be the first choice, all other factors being equal.

2.4.3. Circulating Loops

The possibility of obtaining mass storage using molecular components is intriguing. Early digital computers used circulating loop delay lines as both main and secondary (mass) stores. These were acoustic delay lines of mercury, etc. Modern versions of "circulating loop stores" are the magnetic bubble devices and the charge-coupled devices of modern electronics. There seems to be a possibility that "circulating loop stores" on a molecular scale might be realizable. In particular, as discussed below, solitons can propagate down a single molecular chain at about the speed of sound. Thus such a molecular chain might serve as a circulating loop store for a molecular mass memory. Since the propagation is a mechanical distortion, it is subject to all mechanical (thermal and other) disturbances and so would need to take place in a cooled environment.

2.5. Molecular Components

There are a number of computer components that must be implemented in molecular form if a molecular computer is to be realized.

(1) Wire

(2) Logical Switches

(3) Flip-flops

(4) Random access memory

(5) Mass memory (serial access acceptable)

(6) Input interface, electronic to molecular level

(7) Output interface, molecular to electronic level

There are other components (such as optical image i/o) that, although desirable, are not necessary. We will not consider these components.

2.6. Wire

The function of wire is to interconnect other components by transmitting signals over a distance, roughly the diameter of the molecular computer, (excluding the mass memory). Thus, it will be necessary that this 'wire' reliably transmit signals over a distance of about 10^{-5} meters. If electromagnetic phenomena is employed for signal transmission, this implies a time delay of about $10^{-5} \times 10^{-8}$ *or* 10^{-13} seconds, which is plenty fast compared to the speed of the proposed logic elements (10^{-12} seconds). Propagation phenomena, to be used for transmission of signals, should thus have a propagation velocity comparable to the speed of light, (3×10^{8} m/s).

Four mechanisms for signal transmission have been suggested in the literature. We have not been able to think of any additional ones. The mechanisms are:

(1) Light propagation in channels, fibers, and free space.

(2) Soliton propagation in molecular chains.

(3) Classic (electrical) conduction in molecular chains.

(4) Super conduction (electrical) in molecular chains.

2.6.1. Light Propagation

The use of light as a signal transmission mechanism is an old idea. There seem to be several molecular possibilities for generating and receiving the light signals. For example, Hanson[15] discusses the prospects and problems of utilizing Frenkel excitons in several ways, one of which is to generate controlled phosphorescence and launch a light signal into a thin film optical wave guide.

The main difficulty with all light propagation systems is the inefficiency inherent in converting into and out of the light system. Hansen recognizes this problem but states

> This problem can be circumvented by using light of the optimal wavelength and by having the desired process occur on a faster time scale than the dissipative process.

Hansen recognizes that his general approach, which includes light propagation, is, in his words, "extremely speculative". We agree. In fact it seems to us that light propagation is not the most

attractive option to pursue for realizing 'wire'. We base this conclusion on the lack of any solution to the efficiency problem mentioned above.

2.6.2. Solitons

There are several potential uses of solitons as carriers of signals in the wire, delay lines, switches and i/o conversion devices of molecular computers. Here we will only summarize one aspect of solitons, their use as signals over 'long' ($\sim 10^{-5}$ m) distances. In this context, it is generally assumed that the word 'soliton' refers to the non-linear topological excitation of a long molecular chain such as polyacetylene, the solitons being the bond-alternation domain walls. Soliton propagation, in effect then, a propagation of a configuration (mechanical) change, is limited by the speed of sound in the material, about 10^{-4}c. Over the distances we have in mind for 'wire' (10^{-5} meters) this corresponds to about 10^{-9} seconds. This is far too slow for our purposes. As a result, the use of solitons to carry long-range signals is not attractive, and so will not receive any further attention in this report.

Solitons may, however, be important for conducting signals over *short* distances in other specialized circuits. Thus, solitons cannot be entirely dismissed as signal carriers.

2.6.3. Classic Conduction

In today's computers, wires range in length and materials, as indicated in Table 1. At best resistivity is about 10^{-6} Ohm-cm for wire materials. The resistance of molecular wires can only be crudely estimated, but for a wire of copper, of length 10^{-3} cm, diameter of 10^{-6} cm the resistance is about 1000 Ohms or so, barely within the range of acceptable resistances. Therefore it will be difficult to employ the usual conductors of higher resistivity as shown in Table 1.

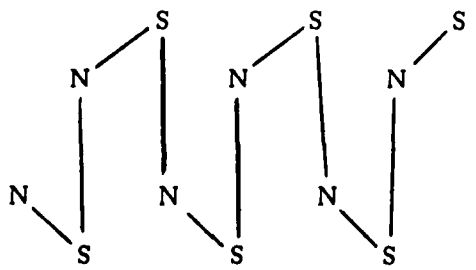
TABLE 1: Properties of 'Wires' Used in Computers					
Material	Resistivity (Ohm-M)	Length Range (M)	Diameter Range (M)	Resistance Range (Ohms)	Utilization
Copper	$1.73 * 10^{-8}$	$10^{-2} - 10^{0}$	$10^{-4} - 10^{-3}$	$10^{-4} - 10^{0}$	Back Planes, PC Boards
Gold	$2.44 * 10^{-8}$	$10^{-5} - 10^{-3}$	$10^{-5} - 10^{-4}$	$10^{-5} - 10^{-1}$	Bonding Wires, I.C.'s
Aluminum	$2.62 * 10^{-8}$	$10^{-6} - 10^{-3}$	$10^{-6} - 10^{-3}$	$10^{-6} - 10^{1}$	Lead Frames, I.C.'s
Doped Si	$1.00 * 10^{-5}$	$10^{-7} - 10^{-4}$	$10^{-6} - 10^{-4}$	$10^{-4} - 10^{3}$	I.C.'s
Silicon	$8.5 * 10^{-4}$	$10^{-7} - 10^{-4}$	$10^{-6} - 10^{-4}$	$10^{-2} - 10^{5}$	I.C.'s

2.6.4. Super Conductors

Because of the difficulties of other possibilities for wires, superconducting wires look attractive. There are several possible molecular superconductors. We will mention only a well known one, $(SN)_x$, polysulfur nitride.

Polysulfur nitride was the first polymeric metal, having been discovered in 1910[16]. It was discovered to be superconducting only in 1975. In the past, there has been some controversy about the transition temperature. It is now thought that the transition temperature for *individual fibers* is 0.28 K but when bundles of non-perfect fibers are measured, the conductivity depends upon interactions between many fibers and the transition temperature then varies from 0.26 to 0.28 K[16].

This material can be formed into single chains[9] of the form:



Actual signal paths would probably employ a number of parallel chains, mostly for reliability. Superconductors seem the best possibility for realizing wires in the molecular computer.

2.7. Logical Switches

The best possibility for a molecule-sized, logical switch (molecular gate) seems to be a an electron tunneling device. According to Carter[9] this idea originated with Pschenichnov who suggested that the transmission coefficient of an electron approaching a finite series of potential barriers would be unity if the electron energy exactly matched the pseudostationary potential of each barrier. Of course any perturbation of any of the barriers would forestall any electron conduction. Thus if a stack of molecules served as the series of potential barriers, then an electrical charge arriving at the edge of one element of the stack would prevent electron conduction from a power source to ground. Since the voltage of such a power supply can be easily controlled, the matching of the electron energy to the stack potentials should be possible. It seems feasible to use $(SN)_x$ as 'wire' between such devices. Figure 1 illustrates this idea to realize a NOR gate.

2.8. Flip-Flops

The best method to implement flip-flops in molecular circuits seems to be the same as in modern digital electronics: Make them from gates.

2.9. Random Access Memory

The creation of a molecular random access memory (RAM) can be approached as it is in today's VLSI circuits, that is , build them from gates. However in the molecular world there is likely to be a much better approach possible by taking advantage of special molecules that can be much more efficient in storing a bit as a charge-controlled, conformational change. This idea is even more speculative than the NOR gate idea discussed above.

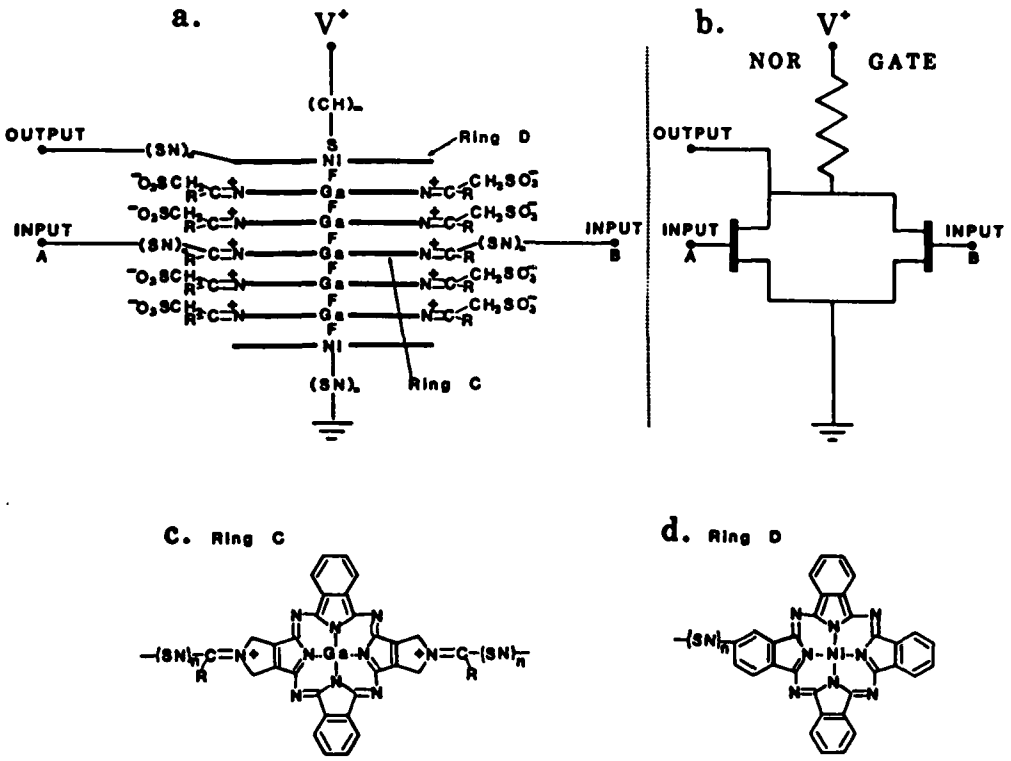


Figure 1. Molecular NOR gate (From Carter, op. cit.).

2.10. Conclusions

The molecular computing ideas presented here are very speculative, but do illustrate that there is considerable potential that molecular computers will be constructed, if not by the exact ideas presented here, then by some other approach.

3. TRANSFORMATION OF METHODS

3.1. Introduction

In computer programming, as in general mathematics, often a method conceived to solve a problem is relatively inefficient compared to what is possible. Consider the calculation of the Discrete Fourier Transform (DFT). Only recently was the Fast Fourier Transform (FFT) discovered, despite the simplicity of the DFT and the large performance improvement offered by the FFT. If we could find a general method of transforming weak methods like the DFT into strong ones like the FFT, a radical improvement of performance would result. Thus, ultimately we seek a general computer program that can accept a weak method and produce a strong one from it. Generally, this is a very difficult and open problem. In this report we will discuss some approaches to this problem and will consider a small, restricted, example of such a program which we developed to aid our understanding of this problem.

3.2. Approach

The general idea if to start with a general, weak, method for solving problems in a domain, we then use the experience in problem solving (and analysis of available acquired-knowledge in the domain) to formulate a stronger solution- finding procedure. Whenever possible, we find specialized and very strong procedures for *subdomains* of the domain.

There is an increasing amount of A.I. research in transforming weak methods into strong ones: Mostow[17], Mitchell[18, 19], Amarel[20, 21], Anzai and Simon[22]. There is similar work in the Soviet Union: (see publication by Glushkov's group in Kiev in mid-seventies on learning strategies for solving planning problems). *Bundy's algebraic manipulation problem* (in the algebraic manipulation system called PRESS)[23] provides a good vehicle for assessing this approach and for identifying key research issues.

Bundy's problem can be formulated as follows:

(1) Consider the domain of R-elementary expressions (given in Bundy, p. 207)[23],

and the axioms of algebra [commutativity, associativity, distributivity, existence of inversion].

(2) Given an equation in the form of an R-elementary expression, in a single unknown, x.

(3) Solve the equation, i.e., find an expression which

(i) is equivalent to the given equation

(ii) has the form

$$x = T[\epsilon x],$$

where the right side is a term from the language of R-elementary expression that does not contain x.

(iii) T is 'simple' (where 'simplicity' is interpreted as *'structural simplicity' of the term*).

Now, the problem can be approached by a *conventional weak method* of *forward search*. This method starts with the given equation and, after successively applying algebraic axioms, transforms the equation into a *form* which is acceptable as a solution.

Bundy's paper focuses on a *stronger method* (which can be viewed as an expert method in this task domain) where a procedure is introduced with a considerable amount of *structure*.

This procedure consists of *five* major methods. These methods are:

Normalize,

Isolate,

Collect,

Attract,

Tidy.

Each of these methods utilizes, in a *specialized manner*, subsets of the algebraic axiom. With each method there is an associated set of *applicability conditions*.

The *Normalize Method* puts the initial equation into a normalized form. Its *applicability condition* is the presence of a non-standard symbol. For example, Normalize converts the expression A/B into A*B^(-1), which "looks" worse, but eliminates the need for the division symbol "/". Thus this form is actually better relative to the total number of symbol types needed to express all equations.

The *Isolate method* strips out connectives from the term that contains x in the expression under consideration, and eventually brings the expression to a form:

$$F : x = T[\epsilon x]$$

The Isolate method is applicable only if the expression F has a *single* occurrence of x.

The *Collect method* reduces the number of occurrences of x in an expression under consideration; of course, this method is applicable only if the number of occurrences of x in an expression F is larger than one. (N(x,F) > 1).

The *Attract method* transforms an expression F so that it has a form on which the collect method can apply. Eventually, the Attract method consists of 'bringing closer' or 'grouping' terms in x. Here again, the method applies only if N(x,F) > 1.

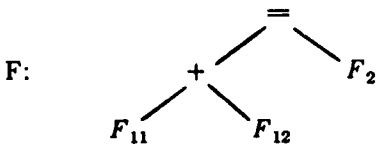
The *Tidy method* reduces the structural complexity of any term. This method is applied wherever possible.

Now, the overall *Solution method* consists off applying in sequence *Normalize, Attract* (if needed), *Collect,* and *Isolate.* Throughout this process, *Tidy* is applied whenever it is applicable.

The interesting issue is understanding the *transition* from the weak, flat, solution method to the structured expert method. What is involved in this transition? What are the bodies of knowledge needed *a priori* to effect the transition? What representation and process can be used to mechanize the transition? What are the key open problems in this area?

Consider a *restricted domain,* with expression from a restricted language, L, where the only 1-ary function symbols are '-', 'square', 'squareroot', and the 2-ary function symbols are '-', '+', *, /. (This means that trigonometric and exponential functions are excluded). The restricted domain permits some simple experiments in the domain.

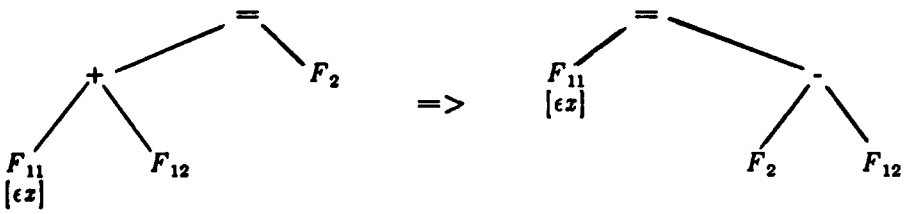
Now consider *applicability conditions* for elements (moves) of the *Isolate method.* The structure of the current expression, F, can be represented in the form of a tree, as follows:



Now let F_1 be the largest subterm of F containing x. We represent F_1 by the subtree of F which is rooted at + . Also, F_{11} is the 'child' of F_1 that contains x, F_{12} is the 'sibling' of F_{11}, F_2 is the 'sibling' of F_1. $N(x,F) = 1$: This is the main applicability condition for the Isolate method; it states that the number of occurrences of x in F is 1.

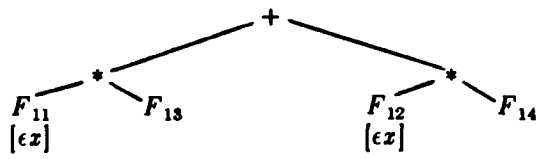
Now, if the conditions apply, then by applying the 'inverse of + ', i.e., by adding $-F_{12}$ to both sides of the equation we obtain a reduction by 1 of the depth of x in F (which is the goal of application of steps in the isolate method).

Thus, the effect of the move can be seen in the form of the transition:



Now, similar rules exist for cases where root-of-F_1 is '-' or '*' or '/'. Slightly different forms are needed where root- of- F_1 is 'squareroot' or 'square'.

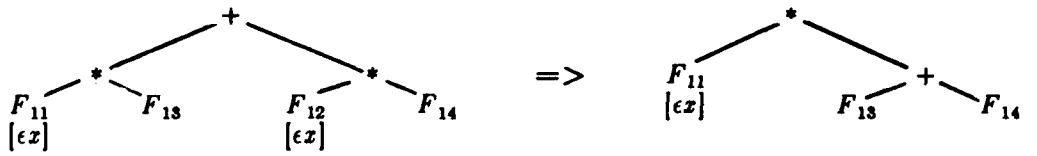
Next, consider what enters in *applicability conditions* for elements (moves) of the *Collect method.* The current expression is F, and F_1 is a subterm (subtree) of F.



F_1 is the least dominating term in x of the expression F, with two of its immediate subterms (immediate subtrees) containing x. In the tree representation, these subterms are shown as F_{11}, F_{12}; and F_{13}, F_{14} stand for terms of arbitrary form. Now, if N(x, F) > 1 : main condition (there are at least 2 occurrences of x) and $F_{11}[\epsilon x] = F_{12}[\epsilon x]$ Then the following *distributive axiom* applies:

$$(u * v) + (u * w) = u * (v + w)$$

The system matches $F_{11}[\epsilon x]$, $F_{11}[\epsilon x]$ to u and the effect of the 'move' application is as follows:



In this transition, the number of occurrences of x decreases -- which is the goal of Collect moves.

Now, the key problem is how to automate the formulation/discovery of applicability conditions for moves such as the above Isolate and Collect moves.

What is involved in approaching the problem in the same way as in *Meta-dendral and LEX[24]?* Briefly, we analyze traces of solutions (obtained via the initial weak method). For each application of a move (i.e., application of an algebraic axiom) we retain the *context of the expression* on which it is applied. Then, we formulate a *generalization* as a pattern which is consistent with all the contexts of expression associated with application of the move. To obtain such generalization we keep a *version space* of generalization.

Upon examination of the Bundy problem, we see that a *LEX-like approach to learning strong applicability conditions for moves is possible* provided that:

(a) We have an 'appropriate' representation framework
for expressing 'expression contexts' (in terms of
which applicability conditions are to be
formulated), and
(b) We have a notion of a 'generalization' hierarchy
to help us formulate patterns in a set of
'expression contexts'.

It appears that there is no difficulty in defining a *generalization hierarchy* in the present domain. The definition of R-elementary expressions (p. 207 Bundy paper) provides a good basis for the hierarchy; but some changes may be necessary. More work is needed here.

To obtain an appropriate representation framework, it is important to reason back from the overall goal of the equation solving task. We can find concepts such as:

-Number of occurrences of x in a term,
-term - subterm relationship,
-term including x,
-notions of structural simplicity of expressions

in the initial problem formulation. Work is needed to assess the difficulty of mechanizing this choice of conceptual framework on the basis of transfer from the problem formulation.

Note that a LEX approach to procedure improvement may be sufficient in this case. However, it may be desirable to have an approach that leads to the structural description of methods presented by Bundy. To do this, we need to be able to discover a global decomposition of *stages* in solution and to formulate for each stage a *micromove/maneuver*. This is a very promising direction of research on expertise acquisition in problem solving. More research is needed here.

3.3. JASON-PRESS System

In order to evaluate the approach taken by Bundy and Welham, in their construction of the PRESS system, we attempted to reconstruct part of PRESS from the fragments that have been published. The part we attempted to reconstruct was just enough to do simple algebra problems. A listing of our version of the PRESS system is given in the appendix.

The reconstruction was largely successful, in that our version could solve many difficult algebra problems and all but one of the examples that PRESS could solve. This is the equation:

$$\left(2^{\cos^2(x)} * 2^{\sin^2(x)} \right)^{\cos(x)} = 2^{(1/4)}$$

We believe with the addition of more specialized axioms this problem could probably be solved by our PRESS system. More interesting however, we have tried some problems that we feel PRESS cannot solve, because of the fundamental structure of PRESS. In particular PRESS has apparently only ad-hoc facilities to factor equations, even when it would be simple to do so. For example our version of *PRESS [JASON-PRESS] cannot solve the simple formula*

$$\log_e^2 x - \log_e x = 0.$$

Factorization is one approach, thus transforming the equation to:

$$y = \log_e x \ ; \ y^2 - y = 0.$$

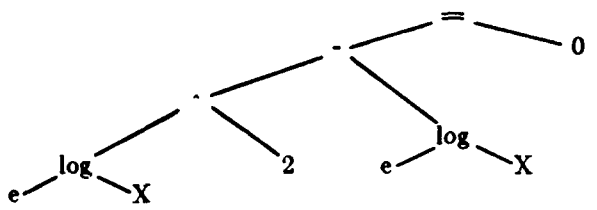
Then again transforming to

$$x = e^y \ ; \ y(y-1) = 0,$$

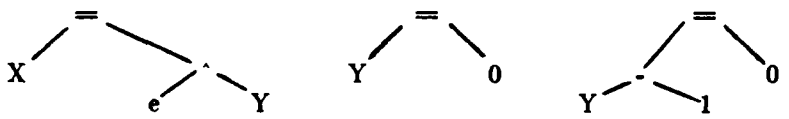
and then again to

$$y = 0 \ ; \ y = 1 \ ; \ x = e^y.$$

This observation led to a new idea to add to the capability of PRESS. This is to add is the maximum height of the covering-tree of X. The reduction method is to reduce the maximum height of this covering-tree of X. For the example shown above, the tree is:



The maximum height of this tree is the length of the far left branch (four edges). The transformed covering-tree (set) is:



The maximum tree-height is now only two links. Although we have not taken the time to implement the FACTOR solution method in JASON-PRESS, it seems clear that it would not only allow a wider range of equations to be solved, but would provide a very clean method of solving the one equation given by Bundy and Welham that our version of PRESS could not solve.

We learned a great deal in implementing and experimenting with our version of PRESS. First, we found the power of Prolog to implement Meta-rules outstanding. Second, the power of the meta-rules to improve performance was impressive. Third, the difficulty of implementing real programming systems is still with us. Finally we gained much insight into the possibility of automatically producing strong programs from weak methods.

3.4. General Strength Improvement Scheme

We now wish to work out a general strength improvement scheme from what we have learned from our experimentation with our PRESS system. PRESS solves equations in ordinary algebra. This algebra has a formal axiomatic definition. Table 2 illustrates a suitable fundamental set of axioms for algebra, and in the form of Prolog facts. It is easily read. For example the entry axiom(5,A+ B,B+ A) says that this is the 5th axiom and that the term A+ B may be

replaced by the term B+ A. A and B may represent any simple or complex expression.

The weak solution method is very simple, but very general. Lemmas are derived from axioms until one is found that will convert a general equation, into the form:

x = Expression (not containing x).

Now we will derive a Prolog representation of this concept.

It is true that we can *solve Expression* with respect to *X* for an *Answer IF* it is true that there is a *lemma* that converts *Expression* to an answer of the form *x = Exp AND* it is true that there is *no-occurrence* of *X* in *Exp*. In Prolog this statement is:

```
solve(Expression, X, Answer) IF
  lemma(Expression, X = Exp) AND
  no-occurrence(X, Exp).
```

The definition of the lemma concept contains the search component. A *lemma* transforms an *Expression* to a *Result IF* there is an *axiom* that relates the *Expression* and the *Result*. In Prolog this is:

```
lemma(Expression, Result) IF axiom(Expression, Result).
```

Also a *lemma* transforms an *Expression* to a *Result IF* there is an *axiom* that transforms *Expression* to the *Answer AND* a *lemma* that transforms the *Answer* to the *Result*. In Prolog this is:

```
lemma(Expression, Result) IF
  axiom(Expression, Answer) AND
  lemma(Answer, Result).
```

The concept of a transformation axiom is a simple statement of fact:

```
axiom (A*B, B*A).
```

There are about two dozen such axioms needed for ordinary algebra. Table 2 illustrates these and the 'structural' axioms to be discussed next.

Table 2. Fundamental axioms.

```
% Equality axioms
axiom(0,A=B,B=A).

% Addition axioms
axiom(1,A+ B,B+ A).    % Commutation
axiom(2,A+ (B+ C),(A+ B)+ C). % Distribution
axiom(3,A+ 0,A).       % Existence of Zero
axiom(4,A+ (-A),0).    % Existence of Negative

% Multiplication axioms
axiom(5,A*B,B*A).      % Commutation
axiom(6,A*(B*C),(A*B)*C).      % Distribution
axiom(7,A*1,A).% Existence of One
axiom(8,A*A^(-1),1) :- A ==0. % Existence of Inverse
axiom(9,A^(-1),1/A).   % Definition

% Distribution
axiom(10,A*B+ A*C,A*(B+ C)).% Add/Mult Distribution

% Exponentiation
axiom(11,A^X*B^X,(A*B)^X).   % Distribution
axiom(12,A^X*A^Y,A^(X+ Y)).  % Distribution
axiom(13,A^1,A).       % Existence
axiom(14,0^0,1).% Definition
axiom(15,log(e^X),X).  % Existence of Inverse
axiom(16,root(X,A^X),A).       % Existence of Inverse
axiom(17,root(2,X),sqrt(X)).   % Definition

% Trig
axiom(18,sin(0),0).    % Definition
axiom(19,sin(pi),0).   % Definition
axiom(20,cos(0), 1).   % Definition
axiom(21,cos(pi),-1).  % Definition
axiom(22,sin(A)^2+ cos(A)^2,1). % Definition
axiom(23,sin(A)*cos(A)^(-1),tan(A)).     % Definition
axiom(24,sec(A)^(-1),sin(A)).    % Definition
axiom(25,cosec(A)^(-1),cos(A)).  % Definition
axiom(26,cotan(A)^(-1),tan(A)).  % Definition

axiom(27,arcsin(sin(A)),A).      % Existence of Inverse
axiom(28,arccos(cos(A)),A).      % Existence of Inverse
axiom(29,arccosec(cosec(A)),A).  % Existence of Inverse
axiom(30,arctan(tan(A)),A).      % Existence of Inverse
axiom(31,arcsec(sec(A)),A).      % Existence of Inverse
axiom(32,arccot(cot(A)),A).      % Existence of Inverse

% Structure
axiom([i,N], E,A) :- axiom(N,A,E).
```

```
axiom(N, E,A) :-
    E=..[Op,L,R],
    ((not atomic(L),axiom(N,L,NL), A=..[Op,NL, R]);
     (not atomic(R),axiom(N,R,NR), A=..[Op,L ,NR])).

axiom([f,Op],A=B,NA=NB) :-
      member(Op,[+ ,-,*,/,^,log,root]),
      (NA=..[Op,A,X],
      NB=..[Op,B,X]);
      member(Op,[-,sqrt,sin,cos,tan,sec,cosec,cotan,
              arcsin,arccos,arctan,arcsec,
              arccosec,arccotan]),
      (NA=..[Op,A],
       NB=..[Op,B]).
```

The first structure axiom is the statement that axioms can be applied in either direction. While some axioms are their own inverse, some are not. Therefore for some axioms, we need to provide an inverse; In Prolog this is:

axiom (x,y) IF axiom (y,x).

Thus either 1)the above statement, can be included in the axiom set, or 2) only the inverses of those axioms to which it applies can be included in the axiom set. For example, axiom(A+ (B+ C), (A+ B)+ C) needs an explicit inverse in the form: axiom((A+ B)+ C,A+ (B+ C)), while axiom(A+ B,B+ A) needs no explicit inverse (it is its own inverse). Table 2 illustrates the use of the first approach.

In the algebra systems we wish to examine, algebraic structures are built up from atomic symbols such as {'3', 'x', 'father'}, and operators such as {=,-,*,/}. Axioms and lemmas apply to both whole expressions and the non-atomic components of the expressions. For example the axiom A+ B <=> B+ A can be applied to x*(a+ b) to yield x*(b+ a). Therefore it is an axiom that an axiom can also be applied to components of an expression. Also then, an *axiom* transforms an *Expression* to a *Result IF* the *Expression* is parsed into an *Operator*, a *Left* part and a *Right* part *AND* either the *Left* part is *not atomic AND* an *axiom* can transform the *Left* part into a *New Left* part *AND* the *Answer* can be constructed *OR* the *Right* part is *not atomic AND* an *axiom* can transform the *Right* part into a *New_Right* part *AND* the *Answer* can be constructed. In Prolog this can be added to our other definitions of axioms as illustrated in Table 2.

In Prolog it is customary to compress the notation and use ':-' in place of IF, ',' in place of AND, and ';' in place of OR. We will switch to this notation below.

3.5. Results

A Prolog query to *solve* a very simple equation $x-a=0$ is: *solve(x-a=0,x,Result).* However for our weak system, the full axiom set is too big for our weak system to explore in a reasonable time. Therefore we have removed all axioms except those that apply to structure, equality, addition and subtraction. The resulting simplified program, but modified to produce a proof, follows.

```
p :-  prf(Proof),
        solve(Proof, x-a=0,x,Result),
        nl,write('Result: '),
        write(Result),nl,
        write('  Proof: '),
        write(Proof),nl.

prf([                   % This is a hint consisting of the first two steps
        [p,l,3],
      [p,l,[p,r,4]]
      | T       ]).

solve(P,E,X,X=A) :- lemma(P,E,X=A),no_occurrence(X,A).   % Solve procedure

lemma([N],E,A) :- axiom(N,E,A).                  % Lemma procedure
lemma([N|M],E,A)  :- lemma(M,E,T),axiom(N,T,A).

no_occurrence(X,[]).
```

```
no_occurrence(X,[|||]).

no_occurrence(X,H)     :- atomic(H), X == H,!.

no_occurrence(X,Exp)   :- Exp =.. [Op, Y|T],
                          no_occurrence(X, Y),!,
                          no_occurrence(X, T),!.

% Fundamental axioms limited to + ,-

% Equality axioms

axiom(0,A=B,B=A).

% Addition axioms

axiom(1,A+ B,B+ A).              % Commutation

axiom(2,A+ (B+ C),(A+ B)+ C).        % Distribution

axiom(i2,(A+ B)+ C,A+ (B+ C)).           % Distribution

axiom(3,A+ 0,A).                 % Existence of Zero

axiom(i3,A,A+ 0).                % Existence of Zero

axiom(4,A+ (-A),0).              % Existence of Negative

axiom(i4,0,A+ (-A)).             % Existence of Negative

axiom(n,(A - B), (A +  (-B)) ).   % Definition of Negative

axiom(in,(A + (-B)), (A - B) ).   % Definition of Negative

axiom(N,E,A) :- axiomp(N,E,A).

axiomp([p,l,N], E,A) :- !, E=..[Op,L,R],
        not atomic(L),axiom(N,L,NL), A=..[Op,NL, R].

axiomp([p,r,N], E,A) :- !, E=..[Op,L,R],
        not atomic(R),axiom(N,R,NR), A=..[Op,L ,NR].

axiom([f,Op],A=B,NA=NB) :-
```

```
        member(Op,[+ ,-]),
        (NA=..[Op,A,X],
         NB=..[Op,B,X]);
        member(Op,[-]),
        (NA=..[Op,A],
         NB=..[Op,B]).

member(X,[H|T]) :- X=H; member(X,T).
```

If a very simple problem such as solve x-a=0 for x is given to the weak method just outlined, it will blindly search through all combinations of its axioms till it finds a result. When we tried this, our VAXll/750 ran four hours without finding a solution. This corresponds to more than a million logical inferences. To verify the correctness of the program, we modified the program to accept hints or even a full proof. Then the system would discover, in a few minutes, a solution if it was given either the first step or the last two steps of the proof. The weak method is indeed weak!

3.6. An Experiment

To explore the potential of the strengthing of this method, we replaced the fundamental axioms, with richer more powerful ones that could be derived from the fundamental set. Again only a restricted set of axioms were included, however besides addition and subtraction, some additional axioms were added. This system follows.

```
test :-
        nl,write(' Attempting to solve :  b*(a-x)=c'),nl,
```

```
        solve(N,b*(a-x)=c,x,A),
        write(' The answer is  : '), write(A),
        write('          Proof: '),write(N),nl.

solve(P,E,X,X=A) :- lemma(P,E,X=A).

lemma([N],E,A) :- ax(N,E,A).
lemma([N|M],E,A)   :- lemma(M,E,T),ax(N,T,A).

ax(N,E,A) :- not atomic(E), axiom(N,E,A).

axiom(1, A=B,B=A).
axiom(2, A*B,B*A).
axiom(3, A+ B,B+ A).
axiom(4 ,(A-B),(-(B)+ A)).
axiom(5 ,A+ B=0,A=(-(B))).
axiom(6, A-B=0,A=B).
axiom(7, A*B=1,A=B^(-1)).
axiom(8, A+ B=C,A=C-B).
axiom(9, A-B=C,A=C+ B).
axiom(10,A*B=C,A=C/B) :- not B = 0.
axiom(11,A*(B-C),A*B-A*C).
axiom(12,A*(B+ C),A*B+ A*C).

axiom(N, E,A) :-
        E=..[Op,L,R],
        ((ax(N,L,NL), A=..[Op,NL, R]);
         (ax(N,R,NR), A=..[Op,L ,NR])).
```

This improved system is much more powerful. It immediately solves, un-aided, the expression x-a=0, and is capable of solving much more complicated expressions such as $b*(a-x)=c$ in just a few seconds. The power of the 'derived' axioms used here is very apparent. In particular the axioms (A-B=0, A=B) was especially useful. It is, of course, just an axiomatization of the results of solving our original example equation. This result, leads to the idea of automatically generating a stronger method from a weak one.

3.7. Generating a Strong Method

We have gained some insight from the above described experiment and our experience with the PRESS system.

(1) Powerful methods require powerful rules (axioms).

(2) Powerful rules can be derived from fundamental ones, that may be weak.

(3) When humans solve equations, they only apply the axioms that will be likely to lead to a solution. The PRESS system mimics this idea with its metarules that control the application of axioms.

Our idea of automatically generating a strong method form a weak one is a generalization of the method used in PRESS. It will also result in a somewhat weaker system than PRESS, as it will not have the extensive, by hand, fine-tuning that has been devoted to the PRESS system. The method follows in the next four subsections.

3.7.1. Metrics

First we define a set of metrics that estimate the distance an expression is from a solution, in each of the corresponding dimensions. These metrics are summarized in Table 3.

TABLE 3: Axiom Sets for PRESS.		
SET NAME	METRIC	EXAMPLE
NORMALIZE*	# of operator symbol types	A/B –> A*B^(-1)
TIDY*	# of total occurrences of all operators	A + 0 –> A
ATTRACT*	Size of the covering tree of X	log(X) + log(X) –> log(X*X)
COLLECT*	# of occurrences of X	X + X –> 2*X
ISOLATE*	The distance of X from the left side of expression	Y + X –> X + Y
FACTOR	Max. depth of the covering tree of X	X*(X-1)=0 –> X=0;X-1=0
* These sets were originally designed by hand by Bundy and Welham for the PRESS system.		

All but the last one were inspired by the PRESS System of Bundy and Welham[23] (The PRESS paper). All the metrics are explained above. For each metric we must write a Prolog procedure to determine if one expression is an improvement over another expression, relative to the metric. For example consider the tidy metric. Its metric is the total number of operators that occur in an expression. A Prolog procedure to measure this *Cost* of an *Expression* is:

```
cost(Expression,0) :- atomic(Expression),!.

cost(Expression,0) :- var(Expression),!.

cost(Expression,Cost) :- Expression =..[Op,R,L],
                         cost(L,Cl), cost(R,Cr),
                         Cost is 1+ Cl+ Cr,!.

cost(Expression,Cost) :- Expression =..[Op,L],
                         cost(L,Cl), Cost is 1+ Cl.
```

The first two lines assign zero cost to atoms and variables. The remainder count operators. The improvement procedure (for tidy) can now be written as:

```
improved(E,R) :- cost(E,Ce), cost(R,Cr), Ce>Cr,!.
```

3.7.2. Selecting Lemmas

Second, We generate lemma for the fundamental axioms as described above, and select only those that can improve an expression more than those that have already been selected. We call the survivors axioms but also signify to which set they belong. Thus axioms generated to tidy expressions and called tidy_axioms. Since the fundamental axioms are bi-directional in their application, the fundamental lemma generation process will itself generate potential expressions for potential improvement. There is a recursive 'boot-strapping' process going on here. It must start somewhere. Therefore a 'starter' axiom must be provided. For the tidy axioms, this will be the null tidy_axiom(E,E).

This technique has some similarity to the Knuth-Bendix procedure[25] and to the 'chunking' ideas of Newell, Rosenbloom and Laird[26, 27].

3.7.3. Structure Axioms Third, we will rewrite the fundamental structure axiom to only employ the new special set of axioms. For the tidy axioms, we will call this procedure 'tidy'. However it must do more than just find some improvement in an expression. We want it to find all possible improvement. Thus it will not only apply axioms to the components of an expression, but to the composite constructions as well. This procedure is very similar to the tidy procedure of PRESS as it appears in the appendix. In Prolog it is:

```
tidy(U,U) :- atomic(U),!.
```

```
tidy(U,U) :- var(U),!.
tidy(U,Z) :- tidyax(U,V),
             ( ( split_exp(V,Op,L,R),
              tidy(L,NL), tidy(R,NR),
              W =..[Op,NL,NR] );
              ( V =.. [Op,R],
              tidy(R,NR),
              W =.. [Op,NR] ) ),
              tidyax(W,Z),
              improved(U,Z),!.

tidy(U,V) :- tidyax(U,V),improved(U,V),!.
tidy(U,U).
```

3.7.4. Arithmetic Fourth, We will provide an axiom to do arithmetic. This will allow expressions such as 1+ 1+ 1 to be reduced to a single number (3 in this case). Obtaining a powerful such a addition reduction of number is not at all simple, if it must be derived from fundamental arithmetic axioms. However it is of course very important for computing and is hard-wired into every modern digital computer. So we will adopt numeric reductions without resorting to fundamental axioms. In Prolog the axiom is:

```
tidyax(Exp,Rslt) :- Exp =..[Op,A1,A2],
                    number(A1),
                    number(A2),
                    Rslt is Exp.
```

In Prolog the 'is' predicate causes expression evaluation. It generates an error message on non numbers, so the check of the arguments was included.

3.7.5. Tidy Axioms

We are now ready to present the Prolog program that will create the tidy_axiom set. We assemble the fundamental set of axioms, the lemma procedure, the 'null' tidy_axiom, the arithmetic reduction tidy_axiom and the tidy procedure derived from the fundamental structure axiom. All of these have been shown above. The procedure that creates the tidy axioms is then:

```
mktidy :- lemma(E,R),
          tidy(R,X),
          tidy(E,W),
          improved(W,X),
          assertz((tidy_axioms(E,X))),fail.
```

The program calls for a fundamental lemma, tidies the result, tidies the original expression using what tidy_axioms it has, and if the tidied lemma result is an improvement, a new tidy_axiom is added to the current set. This program, once started, will not terminate till it runs out of memory space, or is interrupted. It will continuously attempt to create new unique tidy axioms, even if this is impossible. When it was started it ran for hours before it ran out of memory. It generated an interesting set of tidy_axioms all within a few seconds. They follow.

```
tidy_ax(arccotan(X)=arccotan(Y),X=Y).
tidy_ax(arccosec(X)=arccosec(Y),X=Y).
```

```
tidy_ax(arcsec(X)=arcsec(Y),X=Y).

tidy_ax(arctan(X)=arctan(Y),X=Y).

tidy_ax(arccos(X)=arccos(Y),X=Y).

tidy_ax(arcsin(X)=arcsin(Y),X=Y).

tidy_ax(cotan(X)=cotan(Y),X=Y).

tidy_ax(cosec(X)=cosec(Y),X=Y).

tidy_ax(sec(X)=sec(Y),X=Y).

tidy_ax(tan(X)=tan(Y),X=Y).

tidy_ax(cos(X)=cos(Y),X=Y).

tidy_ax(sin(X)=sin(Y),X=Y).

tidy_ax(sqrt(X)=sqrt(Y),X=Y).

tidy_ax(-X=-Y,X=Y).

tidy_ax(root(X,Y)=root(Z,Y),X=Z).

tidy_ax(log(X,Y)=log(Z,Y),X=Z).

tidy_ax(X^Y=Z^Y,X=Z).

tidy_ax(X/Y=Z/Y,X=Z).

tidy_ax(X*Y=Z*Y,X=Z).

tidy_ax(X-Y=Z-Y,X=Z).

tidy_ax(X+ Y=Z+ Y,X=Z).

tidy_ax(arccot(cot(X)),X).

tidy_ax(arcsec(sec(X)),X).

tidy_ax(arctan(tan(X)),X).

tidy_ax(arccosec(cosec(X)),X).

tidy_ax(arccos(cos(X)),X).

tidy_ax(arcsin(sin(X)),X).

tidy_ax(cotan(X)^-1,tan(X)).

tidy_ax(cosec(X)^-1,cos(X)).

tidy_ax(sec(X)^-1,sin(X)).
```

```
tidy_ax(sin(X)*cos(X)^-1,tan(X)).

tidy_ax(sin(X)^2+ cos(X)^2,1).

tidy_ax(cos(pi),-1).

tidy_ax(cos(0),1).

tidy_ax(sin(pi),0).

tidy_ax(sin(0),0).

tidy_ax(root(X,X^X),X).

tidy_ax(log(e^X),X).

tidy_ax(0^0,1).

tidy_ax(X^1,X).

tidy_ax(X^Y*Z^Y,(X*Z)^Y).

tidy_ax(X*Y+ X*Z,X*(Y+ Z)).

tidy_ax(X*X^-1,1).

tidy_ax(X*1,X).

tidy_ax(X+ (-X),0).

tidy_ax(X+ 0,X).

tidy_ax(X=X,true).
```

It can be seen that there are many similarities between these and the tidy axioms of the PRESS system.

3.8. Conclusions

Our primary goal was to examine the possibility of automatically generating a strong method from a weak one. We were not able, in the limited time we had, to investigate all parts of this problem, but we did determine a general methodology. We successfully experimented with generating strong axioms from weak ones.

Our general methodology can be summerized as follows:

(1) Define metrics that estimate the distance to a solution in each dimension of the problem space.

(2) Generate specialized sets of axioms, selected by each metric.

(3) Apply each set on turn to the expression to be solved so the application of each set provides suitable expressions for the next set to be applied. See the PRESS system for details of this.

It does seem feasible to automatically generate strong methods from weak ones in some domains, at least in ordinary algebra. At the moment we do not have such a system, but it seems reasonable to expect such a system could be made to work. Clearly much remains undone.

The potential applications of these ideas include not only algebraic manipulation, but more complex systems as well. For example, if one starts from an axiom set for a Galois field then it may be possible to derive some interesting methods for dealing with cipher problems, etc. Similarily, if complex arithmetic is added, then some interesting methods for solving signal processing problems may be possible.

One view of the method is that it is a 'super-compiler'. It is clear that in the future, the general approach presented here could have significant impact on software development techniques.

4. REFERENCES

1. A. M. Despain, G. J. MacDonald, A. M. Peterson, O. S. Rothaus, and J. F. Vesecky, *Radical Computing,* Jason, McLean, Va (April, 1983). Tech. Rep. JSR-82-701
2. Saul Amarel, Alvin M. Despain, Curtis G. Callan, Jr., and Oscar. S. Rothaus, *Radical Computing II,* Jason, McLean, Va (June 1984). Tech. Rep. JSR-83-701
3. A. M. Despain and Y. N. Patt, "The Aquarius Project," *Digest of Papers, COMPCON Spring 1984,* pp.364-367, IEEE Press (Spring 1984).
4. C. A. Mead and L. A. Conway, in *Introduction to VLSI Systems,* Addison Wesley (1980).
5. R. P. Feynman, "The wonders that await a Micro-Microscope," *The Saturday Review* Vol. **43**, p.45 (April 1960).
6. Forrest L. Carter, "Problems and Prospects of Future Electroactive Polymers and 'Molecular' Electronic Devices," in *NRL Program on Electroactive Polymers, First Annual Report,* ed. L. D. Lockhart Jr., Naval Research Laboratory, Washington D.C. (1979).
7. Forrest L. Carter, "Studies in Tunnelling in Short Periodic Arrays," in *2nd International Workshop on Molecular Electronic Devices,* ed. Forrest L. Carter, to be published (1983).
8. Forrest L. Carter, "The Chemistry in Future Computers," *Proc. 6th Int. Conf. Comput. Chem. Resch Ed.,* p.225, Elsevier (1983).
9. Forrest L. Carter, "Molecular Level Fabrication Techniques and Molecular Electronic Devices," *J. Vac. Sci. Technol.* Vol. **B 1**(4) (1983).
10. Forrest L. Carter, "Toward Computing at the Molecular Level," in *Microelectronics-Structure and Complexity,* ed. Raymond Dingle, Plenum, New York (1983).
11. Forrest L. Carter, "Electron Tunnelling in Short Periodic Arrays," in *Molecular Electronic Devices,* ed. Forrest L. Carter, Marcel Dekker, New York (1982).
12. Forrest L. Carter, "Conformational Switching at the Molecular Level," in *Molecular Electronic Devices,* ed. Forrest L. Carter, Marcel Dekker, New York (1982).
13. Forrest L. Carter, in *Molecular Electronic Devices,* Marcel Dekker, New York (1982).
14. R. P. Feynman, "There is Plenty of Room at the Bottom," pp. 282-296 in *Miniaturization,* ed. H. D. Gilbert, Reinhold, New York (1961).
15. David M. Hanson, "Effects of External Electric Fields and Field Gradients on the Motion of Frenkel Excitons in Molecular Crystals," pp. 109 in *Molecular Electronic Devices,* ed. Forrest L. Carter, Marcel Dekker, New York (1982).
16. M. M. Labes, P. Love, and L. F. Nichols, "Polysulfur Nitride - A Metallic, Superconducting Polymer," *Chem. Rev.* Vol. **79**(1), pp.1-15 (January 1979).
17. D. J. Mostow, "Mechanical Transformation of Task Heuristics into Operational Procedures," *Ph D Thesis,* Carnegie Mellon University (April 1981).
18. T. M. Mitchell, P. E. Utgoff, and R. Banerji, "Learning by Experimentation: Acquiring and Refining Problem Solving Heuristics," in *Machine Learning,* ed. Mitchell, Tioga (1982).
19. T. M. Mitchell, "Learning and Problem Solving," *Proc. IJCAI-83,* Morgan Kaufmann (1983).
20. Saul Amarel, "Program Synthesis as a Theory Formation Task - Problem Representations and Solution Methods," in *Machine Learning: An Artificial Intelligence Approach, Vol II.,* ed. Michalski, Morgan Kaufmann, Palo Alto CA (1985).
21. Saul Amarel, "Problems of Representation in Heuristic Problem Solving: Related Issues in the Development of of Expert Systems," in *Methods of Heuristics,* ed. Bischoff, Lawrence Erlbaum (1983).

22. Y. Anzai and H. A. Simon, "The Theory of Learning by Doing," *Psychological Review* Vol. 86 (1979).

23. A. Bundy and B. Welham, "Using Meta-Level Inference for Selective Application of Multiple Rewrite Rule Sets in Algebraic Manipulation," *A.I Journal*(16) (1981).

24. T. M. Mitchell, "Version Spaces: An Approach to Concept Learning," *Ph D Thesis*, Stanford University (1978).

25. D. E. Knuth and P. Bendix, "Simole Word Problems in Universal Algebras," pp. 263-297 in *Computational Problems in Abstract Algebras*, ed. J. Leech, Pergamon (1970).

26. P. S. Rosenbloom and A. Newell, "Learning by Chunking: Summary of a Task and a Model," *Proc. AAAI-82 Natl. Conf. on AI*, Morgan Kaufmann (1982).

27. J. E. Laird, P. S. Rosenbloom, and A. Newell, "Towards Chunking as a General Learning Mechanism," Tech Report, Computer Science, Carnegie Mellon University (March 1984).

5. APPENDIXES

APPENDIX A-1

PRESS

```
% Main PRESS system                    Despain  August 84

 :- consult(normalize),
    consult(solve),
    reconsult(display).

% Test of press...
tp :- reconsult(press_tests),
      nl, t(N, Exp), write('CASE: '),
      write(N),nl, press(Exp, x, Ans),
      display(Exp, Ans), fail.

% Main PRESS procedure.

press(Exp, X, Ans) :- tidy( Exp,E), normalize(E, Rslt), solve(Rslt, X, Ans),!.
```

```
% Normalize Procedure.          Despain, November 1984.


normalize(U,V) :- removeall(U, W),!,restrictall(S, W, V),!.

removeall(U,Z) :- remove(U,V), V =.. [Op|Args],
                  remove_list(Args,Nargs),
                  W =.. [Op|Nargs],
                  remove(W,Z).
removeall(U,V) :- remove(U,V).

remove_list([],[]).
remove_list([H|T],[Nh|Nt]) :- removeall(H,Nh),remove_list(T,Nt).

% Remove axioms
remove(A<B, B>A).
remove(A/B, A*B^(-1)).
remove(root(R, X), X^R^(-1)).
remove(A, A).


% Restrict procedure

restrictall(S, U, Z) :- restrict(S, U, V),
                        V =.. [Op|Args],
                        restrict_list(S, Args, Nargs),
                        W =.. [Op|Nargs],
                        restrict(S, W, Z).
restrictall(S, U, V) :- restrict(S, U, V).

restrict_list(S, [], []).
restrict_list(S, [H|T],[Nh|Nt]) :- restrictall(S, H, Nh),
                                   restrict_list(S, T, Nt).


% Restrict axioms
restrict(uminus, -(U+ V),(-U)+ (-V)).
restrict(_, U, U).

% Test the normalize Procedure.          Despain, November 1984.

t :- reconsult(normalize), reconsult(display), tl, td, tu.

tl :- Exp = (0<(-(a+ b))),
     normalize(Exp, Ans), display(Exp, Ans).

td :- Exp = a+ b/c,
     normalize(Exp, Ans), display(Exp, Ans).

tu :- Exp = -(a+ b),
     normalize(Exp, Ans), display(Exp, Ans).
```

```
% Tidy procedure.          Despain, July 1984.

tidy(U,Z) :- tidyax(U,V), V =.. [Op|Args],tidy_list(Args,Nargs),
        W =.. [Op|Nargs],tidyax(W,Z),!.
tidy(U,V) :- tidyax(U,V).

tidy_list([],[]).
tidy_list([H|T],[Nh|Nt]) :- tidy(H,Nh),tidy_list(T,Nt).


% Tidy axioms
tidyax(U = U,true).
tidyax(-(-U),U).

tidyax(U^1,U).
tidyax(1^U,1).
tidyax(U^0,1).
tidyax(0^U,0) :- number(U), U == 0.

tidyax(U*1,U).
tidyax(1*U,U).
tidyax(U*0,0).
tidyax(0*U,0).

tidyax(U+ 0,U).
tidyax(0+ U,U).

tidyax(U-0,U).
tidyax(0-U,-U).

tidyax(0+ (-U),-U).

tidyax(U-(-V),U+ (V)).

tidyax(log(U,U),1).
tidyax(log(U,1),0).

tidyax(U*(V*W),U*V*W)  :- atomic(V).
tidyax((U*V)*W,U*V*W) :- atomic(W).

% This axiom does simple arithmetic when it can to tidy up expressions.
tidyax(Exp,Rslt) :- Exp =..[Op,A1,A2],
                    number(A1),
                    number(A2),
                    Rslt is Exp.

% These may not be a tidy axioms that are needed to solve eqs,
% but collect axioms.
tidyax(X * X, X^2).
tidyax(X * X^N, X^M) :- M is N+ 1.
tidyax(X^N * X, X^M) :- M is N+ 1.
tidyax(X^N * (X), X^M) :- M is N+ 1.
tidyax(X^L * X^N, X^M) :- M is N+ L.

% These may not be tidy axioms, but isolate axioms.
```

```
%tidyax(A - B = 0, A = B).
%tidyax(A + B = 0, A = - B).
%tidyax(A * B^(-1) = 1, A = B).
%tidyax(A * B^(-1) = 1, A = B).

% These are most likely attract axioms.
%tidyax(U^A = U^B , A = B).
%tidyax(U^A = U , A = 1).
%tidyax(U^A - U^B = 0 , A = B).
%tidyax(U^A - U = 0 , A = 1).

tidyax(U,U).
% Test of tidy procedure.          Despain, July 1984.

t :- reconsult(tidy),
     Exp = (((2*x)^0)^((2*x)^1) = (1^(2*x))*cos(x)^0),
     tidy(Exp, Rslt),
     print(' Input  Expression is : '),
     print(Exp), nl,
     print(' Tidied Expression is : '),        % Should print 'true'.
     print(Rslt), nl.
```

```
% Test case.

% Tidy axioms
tidyax(U = U,true).
tidyax(-(-U),U).
tidyax(U^1,U).
tidyax(1^U,1).
tidyax(U^0,1).
tidyax(0^U,0) :- integer(U), U == 0.
tidyax(U*1,U).
tidyax(1*U,U).
tidyax(U*0,0).
tidyax(0*U,0).
tidyax(U+ 0,U).
tidyax(0+ U,0).
tidyax(U-0,U).
tidyax(0-U,(-U)).
tidyax(U+ (-0),U).
tidyax(0+ (-U),(-U)).
% It is not clear if the following axiom is a true tidy axiom or not....
tidyax(U-(-V),U+ V).

tidyax(U,U).
```

```
% Isolate procedure.                    Despain    Nov 84
% isolate(Position, Exp, Ans).
:- consult(isolax),
   reconsult(tidy),
   reconsult(display).

% Test.
  ti :- reconsult(isolate_test),!,nl, t(3),fail ; true.

isolate([1|T], L = R, A) :- iso(T, L = R, A).
isolate([2|T], L = R, A) :- iso(T, R = L, A).

%iso(U,V,X) :- write(U),write(' : '),write(V),write(' : '),write(X),nl,fail.
% this is for testing...

iso([], X, X).
iso([H|T], FX = W, Ans) :- !,
        isolax(H,FX = W, RHS, Condition),
        Condition,
        tidy(RHS, New),!,
        iso(T, New, Ans).

% Misc functions
arbint(n0).
even(N) :- integer(N), 0 is N mod 2.
odd(N) :- integer(N), 1 is N mod 2.
```

```
% Isolate procedure testing.                    Despain    Nov 84

t(1) :- Exp = (-sin(x) = y),
        isolate([1], Exp, Ans),
        display(Exp, Ans).

t(2) :- Exp = (sin(a)+ x=z),
        isolate([2], Exp, Ans),
        display(Exp, Ans).

t(3) :- Exp = (log(e,x^2-1)=3),
        isolate([1,2,1,1], Exp, Ans),
        display(Exp, Ans).

t(4) :- Exp = (x*3+ y=z),
        isolate([1,1], Exp, Ans),
        display(Exp, Ans).

t(5) :- Exp = (x^3=z),
        isolate([1], Exp, Ans),
        display(Exp, Ans).

t(6) :- Exp = (x^4+ y=z),
        isolate([1,1], Exp, Ans),
        display(Exp, Ans).

t(7) :- Exp = ((a+ x^6)*y=z),
        isolate([1,2,1], Exp, Ans),
        display(Exp, Ans).

t(8) :- Exp = (x^y=z),
        isolate([1], Exp, Ans),
        display(Exp, Ans).
```

```
%Solve Procedure                                Despain        Nov 84
  :- reconsult(tidy),      reconsult(isolate), reconsult(singleocc),
     reconsult(no_occur), reconsult(position),
     reconsult(collect),  reconsult(attract),  reconsult(closeness).

% Solve procedure testing.                      Despain    Nov 84
ts :- reconsult(display), reconsult(solve_tests),
      nl, t(N, Exp),
      write('CASE: '), write(N),nl, solve(Exp, x, Ans),
      display(Exp, Ans), fail.

% Main solve procedure.

solve(L = R, X, Ans) :-
        singleocc(X, L = R),!,
        position(X, L = R, P),!,
        isolate(P, L = R, Ans).

solve(Eqn, X, Ans) :-
        collect(X, Eqn, New),!,
        solve(New, X, Ans).

solve(Eqn, X, Ans) :-
        attract(X, Eqn, New),
        closeness(X, Eqn, EC),
        closeness(X, New, NC),
        EC > NC, !,
        solve(New, X, Ans).
% Solve procedure testing.                      Despain    Nov 84

  :- consult(tidy),      consult(isolate),  consult(solve),
     consult(singleocc), consult(no_occur), consult(position),
     consult(display).

t :- nl, t(N, Exp), write('CASE: '), write(N),nl, solve(Exp, x, Ans),
     display(Exp, Ans), fail.

t(1,(-sin(x) = y)).
t(2,(sin(a)+ x=z)).
t(3,(sin(a)+ x=z)).
t(4,(x*3+ y=z)).
t(5,(x^3=z)).
t(6,(x^4+ y=z)).
t(7,((a+ x^6)*y=z)).
t(8,(x^y=z)).
```

```
% Singleocc procedure

singleocc(X,[H|T]) :- singleocc(X, H),!,
                      no_occurence(X, T).
singleocc(X,[H|T]) :- no_occurence(X, H),!,
                      singleocc(X, T).
singleocc(X,X).
singleocc(X,Exp) :- Exp =.. [Op, Y|T],
          singleocc(X, Y),
          no_occurence(X, T).
singleocc(X,Exp) :- Exp =.. [Op, Y|T],
          singleocc(X, T),
          no_occurence(X, Y).

% Test of singleocc procedure

t :- reconsult(singleocc), reconsult(no_occur),tt.

tt :- singleocc(x, [g(y,a+ b,[e,x,d]),c]).
tf :- singleocc(x, g(y,a+ b,x),x).
tn :- singleocc(x, g(y,a+ b,z),c).

singleocc(X,[H|T]) :- singleocc(X, H),!,
                      no_occurence(X, T).
singleocc(X,[H|T]) :- no_occurence(X, H),!,
                      singleocc(X, T).
singleocc(X,X).
singleocc(X,Exp) :- Exp =.. [Op, Y|T],
          singleocc(X, Y),
          no_occurence(T, Y).
singleocc(X,Exp) :- Exp =.. [Op, Y|T],
          singleocc(X, T),
          no_occurence(X, Y).
% no_occurence procedure
no_occurence(X,[]).
no_occurence(X,H)     :- atomic(H), X == H.
no_occurence(X,[H|T]) :- no_occurence(X, H),!,
                         no_occurence(X, T).
no_occurence(X,Exp)   :- Exp =.. [Op, Y|T],
                         no_occurence(X, Y),!,
                         no_occurence(X, T).
```

```
% Position procedure.                    Despain, July 1984.

% Singleocc procedure must proceed this, as it will wrongly report if
% there is no occurance of X.

% Position(Symbol, Expression, Index).

position(X,[],[]).
position(X,[X],[1]).
position(X,[X|T],[1])   :-  no_occurence(X, T).
position(X,[H|T],[1|Q]) :-  no_occurence(X, T), position(X,H,Q).
position(X,[H|T],[P|Q]) :-  no_occurence(X, H), position(X,T,[S|Q]),
                            P is S + 1.

position(X,X,[1]).
% position(X,X,[]). THIS ORIGINAL DEFINITION SEEMS WRONG

position(X,Exp,Pos)   :- Exp =.. [Op|List],
                         Op == .,!,
                         position(X,List,Pos).
% Test of Position procedure.                    Despain, July 1984.


nox(Exp) :- no_occurence(x,Exp).
po(Exp) :- position(x,Exp,N),print(N).

t :- reconsult(no_occur), reconsult(position), reconsult(display),!,
    nl, t(N, Exp), write('CASE: '), write(N),nl,position(Exp,Ans),
    display(Exp, Ans), fail.

position(Exp, Ans) :- position(x, Exp, Ans),!.

t(false,[x,[a,[x]],c]).

t(true,[y,[a,[x]],c]).

t(1,(-sin(x) = y)).

t(2,(sin(a)+ x=z)).

t(3,(sin(a)+ x=z)).

t(4,(x*3+ y=z)).

t(5,(x^3=z)).

t(6,(x^4+ y=z)).

t(7,((a+ x^6)*y=z)).

t(8,(x^y=z)).

t(none, [y,[a,[w]],c]) :-  write('Note the false answer here'),nl.
```

```
display(Exp, Ans) :-
    print(' Input Expression is : '), print(Exp),nl,
    print(' The  Solution  is   : '), print(Ans),nl,nl.


% Test Equations for testing PRESS procedures.  Despain  August 84

% Test expressions: t(Name, Expression).
t(0,(x+ x=a-b)).
t(1,(-sin(x) = y)).
t(2,(cos(a)+ x=z)).
t(3,(log(e,x^2-1)=3)).
t(4,(x*3+ y=z)).
t(5,(x^3=z)).
t(6,(x^4+ y=z)).
t(7,((a+ x^6)*y=z)).
t(8,(x^y=z)).

% Tests of Bundy and Welham.
t(9,  (log(e,(x+ 1)*(x-1))=3)).
t(10, (log(e,x+ 1)+ log(e,x-1)=3)).
t(11, (2^(x^2)^(x^3)=2)).
t(12, ((2^(cos(x)^2)*2^(sin(x)^2))^cos(x) = 2^(1/4))).

% Collect procedure                              Nov 84

% Warning, dont try to backtrack through this procedure.

:-  reconsult(tidy),
    reconsult(contains),
    reconsult(match),
    reconsult(collax),
    reconsult(least_dom),
    reconsult(no_occur),
    reconsult(display).

tc :- reconsult(collect_tests), nl,
     t(N, Exp),
     write('CASE: '),
     write(N),nl,
     write('Input: '),
     write(Exp),nl,
     collectx(x, Exp, Ans),
     display(Exp, Ans), fail.

 tc11 :- t(11,E), collect(x,E,A), nl,display(E,A),nl.
 tc3  :- t(3 ,E), collect(x,E,A), nl,display(E,A),nl.

collectx(X, Old, New) :-  collect(X, Old, New), !.

collect(X, Old, New) :-
         contains(X, Old),
         least_dom(X, Old),
         collax(U, LHS, RHS),
         match(Old, LHS),
```

```
        contains(X, U),
        tidy(RHS, New).

collect(X, Old, New) :-
        contains(X, Old),
        Old =..[Op,AH|AT],
        ( ( ( collect(X, AH, NH),
              New =..[Op, NH|AT] );
            ( collect(X, AT, NT),
              New =..[Op, AH|NT] ) );

          ( AH =..[Op, H|[T3]],        % This solves the problem created by the
            T =.. [Op,T3|AT],          % fact that expressions are scanned
              ( collect(X, T, NT),     % backward when Op's are the same.
                New =..[Op, H,NT] );     % Ex: x*2*x*x is collected..
              ( collect(X, H, NH),
                New =..[Op, NH|T] ) ) ).

%         ( AT =..[Op, H1|T1],         % It is not clear that this case ever
%           H =.. [Op, AH|H1],         % occurs, so it may be useless work.
%           ( collect(X, H, NH),
%             New =..[Op, NH|T1] );
%           ( collect(X, T1, NT),
%             New =..[Op, H|NT] ) ) ).
```

```
% Test of collect procedure.                    Despain   August 84
tc :- t(10,E), collect(x,E,A), nl,write(A),nl.

t :- reconsult(collect), reconsult(tidy),
     reconsult(contains), reconsult(match),reconsult(collax),
     reconsult(least_dom),
%    reconsult(test_eqns),
     reconsult(display),!,
     nl, t(N, Exp), write('CASE: '), write(N),nl, collectx(x, Exp, Ans),
     display(Exp, Ans), fail.

collectx(X, Old, New) :- collect(X, Old, New), !.

% Test Equations for testing Collect procedure.  Despain   August 84

% Test expressions: t(Name, Expression).
t(0,(x+ x=a-b)).
t(1,(-sin(x) = x)).

% Tests of Bundy and Welham.
t(10, (log(e,x+ 1)+ log(e,x-1)=3)).
t(9,  (log(e,(x+ 1)*(x-1))=3)).
t(11, (2^(x^2)^(x^3)=2)).
t(12, ((2^(cos(x)^2)*2^(sin(x)^2))^cos(x) = 2^(1/4))).

% Collect axioms.

% special collect axiom...
collax(X, X^X^0 = 1, X = 1).
collax(X, X^X^N = 1, (X = 0 ; X = 1)).

collax(X, X - X, 0).
collax(X, X + X, 2*X).

collax(X, X * X, X^2).
collax(X, X+ W*X, X*(1+ W)).
collax(X, X*U+ X*V, X*(U+ V)).

collax(X, X*X^V, X^(1+ V)).
collax(X, X^U*X^V, X^(U+ V)).
collax(X, U^X*V^X, (U*V)^X).
collax(X, (X+ U)*(X-U), X^2-U^2).

b0.
collax(X, log(X,U)+ log(X,V), log(X,U*V)).
collax(X, log(X,U)+ log(X,V), log(X,U*V)) :-b0.
collax(X, log(X,U)-log(X,V), log(X,U*V^(-1))).
collax(X, log(U,X)+ log(V,X), log(E,X)*(log(U,E)+ log(V,E))).

collax(X, sin(X)^2+ cos(X)^2,1).
collax(X, sin(X)*cos(X)^(-1),tan(X)).
collax(X, sin(X)*cos(X),sin(2*X)*2^(-1)).
```

```
% Least_dominate procedure.

least_dom(X, X^Y = 1) :- !.    %Special case for x^x^N=1 etc....
least_dom(X, Exp) :-
        Exp =..[Op,H|T],
        contains(X, H),
        contains(X, T).
% Contains procedure.  contains(X, Exp).

contains(X, X).
contains(X, Exp) :-
        Exp == [[]],
        Exp =..[Op,H|T],
        (contains(X, H);
         contains(X, T)).
% Match procedure.

match(A, A).
match(A = B, X = Y) :- crossmatch(A, B, X, Y).
match(A * B, X * Y) :- crossmatch(A, B, X, Y).
match(A + B, X + Y) :- crossmatch(A, B, X, Y).
match(A - B, (-Y) + X ) :- crossmatch(A, B, X, Y),!.
match((-A) + B, Y - (X)) :- crossmatch(A, B, X, Y),!.
match(U, V) :-
        U == [[]],
        U =..[Op,H|T],
        V == [[]],
        V =..[Op,F|L],
        match(H, F),
        match(T, L).
match(U^1, U).
match(U, U^1).

crossmatch(A, B, X, Y) :-
        ((match(A, X),
          match(B, Y));
         (match(A, Y),
          match(B, X))).
```

```
% Attract procedure

 :- reconsult(attrax),
    reconsult(least_dom),
    reconsult(match),
    reconsult(contains),
    reconsult(closeness),
    reconsult(tidy),
    reconsult(display).

% Test of attract procedure
ta :- reconsult(attract_tests),
     nl, t(N, Exp),
     write('CASE: '),
     write(N),nl,
     attractx(x, Exp, Ans),
     display(Exp, Ans),fail.

ta22 :- t(22, Exp),attract(x, Exp, Ans),
       nl,write('ta22 Input:  '), write(Exp),nl,
       nl,write('ta22 Output: '), write(Ans),nl.

t(22, (2^((x^2)^(x^3))=2^x^0)).

ta21 :- t(21, Exp),attract(x, Exp, X),
       nl,write('ta21 Input:  '), write(X),nl,
       nl,write('ta21 Output: '), write(X),nl.

t(21, (2^((x^2)^(x^3))=2)).

ta23 :- t(23, Exp),attract(x, Exp, X),
       nl,write('ta21 Input:  '), write(X),nl,
       nl,write('ta21 Output: '), write(X),nl.

t(23, (x*2*x*x=2)).


attractx(X, Old, New) :- attract(X, Old, New), write('X : '), write(New),nl, !.

attract(X, Old, New) :-
         least_dom(X, Old),
         attrax(Ulist, LHS, RHS),
         match(Old, LHS),
         checklist_contains(X, Ulist),
%        closeness(X, LHS, Sold),
%        closeness(X, RHS, Snew),
%        Snew < Sold ,
        !,
         tidy(RHS,New).

attract(X, Old, New) :-
         Old == [],
         Old == [[]],
         Old =..[Op,H|T],
```

```
        ((attract(X, H, NH),
        New =..[Op,NH|T]);
        (attract(X, T, NT),
        New =..[Op,H|NT])).

attract(X, L = R , New) :-
        least_dom(X, Old),
        attras(Ulist, L = R, RHS),
        match(Old, LHS),
        checklist_contains(X, Ulist),!,
        tidy(RHS,New).

attras([U,V],U^V = 1, U = 1).

%checklist_contains(X, [H|T]) :- contains(X,H).
%checklist_contains(X, [H|T]) :- checklist_contains(X, T).

checklist_contains(X, [H]) :- contains(X,H).
checklist_contains(X, [H|T]) :- contains(X,H), checklist_contains(X, T).
% Closeness procedure.
tcl :- reconsult(least_dom),
      reconsult(contains),
      closeness(x,a+ x=b*(c+ x),M),
      nl, write(M),nl.

closeness(X, E, V) :- close(X, E, V),!.
close(X, X, 0).

close(X, Exp, M) :-
        least_dom(X, Exp),
        tree_size(X, Exp, _, 0, M),!.

close(X, Exp, M) :-
        Exp == [],
        Exp == [[]],
        Exp =..[Op,H|T],
        ((contains(X, H), close(X, H, M));
         (contains(X, T), close(X, T, M))).

tree_size(X,  [] , 0, Size_in, Size_in)  :- !.
tree_size(X, [[]], 0, Size_in, Size_in)  :- !.
%tree_size(X,   X,  1, Size_in, Size_in)  :- !.
tree_size(X,   X,  1, Size_in, Size_out) :- Size_out is Size_in + 1,!.
tree_size(X,   Y,  0, Size_in, Size_in ) :- atomic(Y),!.
tree_size(X, Exp, HX, Size_in, Size_out) :-
        Exp == [],
        Exp == [[]],
        Exp =..[Op,H|T],
        tree_size(X, H, FX, Size_in, S),
        tree_size(X, T, GX, S, S2),
        or(FX, GX, HX),
      ((Op = '.' ) ->
         (Size_out is S2) ;
         (Size_out is S2 + HX)).
```

```
or(1,_,1).
or(_,1,1).
or(0,0,0).
```

```
% Fundamental axioms. Suggested by Rothaus.      Despain, July 84.

% Addition axioms
axiom(A+ B,B+ A).                % Commutation
axiom(A+ (B+ C),(A+ B)+ C).      % Distribution
axiom(A+ 0,A).                   % Existence of Zero
axiom(A+ (-A),0).                % Existence of Negative


% Multiplication axioms
axiom(A*B,B*A).                  % Commutation
axiom(A*(B*C),(A*B)*C).          % Distribution
axiom(A*1,A).                    % Existence of One
axiom(A*A^(-1),1) :- A ==0.      % Existence of Inverse
axiom(A^(-1),1/A).               % Definition

% Distribution
axiom(A*B+ A*C,A*(B+ C)).        % Add/Mult Distribution

% Exponention
axiom(A^X*B^X,(A*B)^X).          % Distribution
axiom(A^X*A^Y,A^(X+ Y)).         % Distribution
axiom(A^1,A).                    % Existence
axiom(0^0,1).                    % Definition
axiom(log(e^X),X).               % Existence of Inverse
axiom(root(X,A^X),A).            % Existence of Inverse
axiom(root(2,X),sqrt(X)).        % Definition

% Trig
axiom(sin(0),0).                 % Definition
axiom(sin(pi),0).                % Definition
axiom(cos(0), 1).                % Definition
axiom(cos(pi),-1).               % Definition
axiom(sin(A)^2+ cos(A)^2,1).     % Definition
axiom(sin(A)*cos(A)^(-1),tan(A)). % Definition
axiom(sec(A)^(-1),sin(A)).       % Definition
axiom(cosec(A)^(-1),cos(A)).     % Definition
axiom(cotan(A)^(-1),tan(A)).     % Definition

axiom(arcsin(sin(A)),A).         % Existence of Inverse
axiom(arccos(cos(A)),A).         % Existence of Inverse
axiom(arccosec(cosec(A)),A).     % Existence of Inverse
axiom(arctan(tan(A)),A).         % Existence of Inverse
axiom(arcsec(sec(A)),A).         % Existence of Inverse
axiom(arccot(cot(A)),A).         % Existence of Inverse
```

```
% This procedure partially compiles axioms at the 1th level

la :- listing(ax).

p_ax(N) :- M is N+ 1,
        ax(N,Exp,Res),
        asserta((ax(M,Exp,Res))), fail.
p_ax(N) :- demolish((ax(N,Exp,Res) :- Body)).
p_ax(N) :- listing(ax).

demolish(Clause) :- retract(Clause),demolish(Clause).



ax(0,Exp,Res) :-  axiom(Exp,Res), improved(Exp,Res).
ax(0,Exp,Res) :-  axiom(Res,Exp), improved(Exp,Res).

ax(N,Exp,Res) :- M is N-1,M>0,
              lemma(Exp,Nex),
              ax(M,Nex,Res) ,
              improved(Exp,Res),
              Exp == Res,          % Put in to fix a bug. Occur check?
              not(axm(M,Exp,Res)).

axm(M,Exp,Res) :-  ax(M,Exp,Res).
axm(M,Exp,Res) :-  N is M-1, N>0, axm(N,Exp,Res).

%-Cost of expression evaluator.
cost(X,0) :- atomic(X),!.
cost(X,0) :- var(X),!.
cost(Exp, Cost) :- Exp =.. [Op|Args], cst(Op,C1),
                list_cost(Args,C2), Cost is C1 +  C2,!.

list_cost([], 0).
list_cost([H|T], Cost) :- cost(H,C1), list_cost(T, C2), Cost is C1 +  C2.


cst(=,1) :- !.
cst(+ ,1) :- !.
cst(-,1) :- !.
cst(*,1) :- !.
cst(^,1) :- !.
cst(/,1) :- !.
cst(_,0).

%lemma(Exp,Res) :- axiom(Res, Exp).
lemma(Exp,Res) :-     Exp == Res,          % Put in to fix a bug. Occur check?
                  axiom(Nex, Exp),
                  Nex =.. [Op|Args],
                  list_axioms(Args,Nargs),
                  Res =.. [Op|Nargs].

list_axioms([],[]).
list_axioms([H|T],[NH|NT]) :- H == NH,     % Put in to fix a bug. Occur check?
                        axiom(H,NH),
```

```
                    list_axioms(T,NT).

improved(Exp,Res) :- cost(Exp,C1),
                     cost(Res,C2),
                     C1>C2,!.

axiom(A,A).

% Addition axioms
axiom(A+ B,B+ A).              % Commutation
axiom(A+ (B+ C),(A+ B)+ C).    % Distribution
axiom(A+ 0,A).                 % Existence of Zero
axiom(A+ (-A),0).              % Existence of Negative


% Multiplication axioms
axiom(A*B,B*A).                % Commutation
axiom(A*(B*C),(A*B)*C).        % Distribution
axiom(A*1,A).                  % Existence of One
axiom(A*A^(-1),1) :- A ==0.    % Existence of Inverse
axiom(A^(-1),1/A).             % Definition

% Distribution
axiom(A*B+ A*C,A*(B+ C)).      % Add/Mult Distribution

% Exponention
axiom(A^X*B^X,(A*B)^X).        % Distribution
axiom(A^X*A^Y,A^(X+ Y)).       % Distribution
axiom(A^1,A).                  % Existence
axiom(0^0,1).                  % Definition
```

DISTRIBUTION LIST

Mr. Saul Amarel
Director
DARPA/IPTO
1400 Wilson Blvd.
Arlington, VA 22209

Dr. Marv Atkins
Deputy Director, Science & Tech.
Defense Nuclear Agency
Washington, D.C. 20305

National Security Agency [2]
Attn R5: Dr. N. Addison Ball
Ft. George G. Meade, MD 20755

Dr. Charles Buffalano
DARPA
Acting Director
1400 Wilson Boulevard
Arlington, VA 22209

Dr. Curtis G. Callan, Jr.
Department of Physics
Princeton University
Box 708
Princeton, NJ 08544

Mr. John Darrah
Sr. Scientist and Technical
Advisor
HQ Space Cmd/XPN
Peterson AFB, CO 80914

Dr. Roger F. Dashen
Institute for Advanced Study
Princeton, NJ 08540

Defense Technical Information [2]
Center
Cameron Station
Alexandria, VA 22314

Dr. Alvin M. Despain
1192 Grizzly Peak Boulevard
Berkeley, CA 94708

CAPT Craig E. Dorman
Department of the Navy, OP-095T
The Pentagon, Room 5D576
Washington, D.C. 20350

CDR Timothy Dugan [2]
NFOIO Detachment, Suitland
4301 Suitland Road
Washington, D.C. 20390

Mr. John Entzminger
Director
DARPA/TTO
1400 Wilson Blvd.
Arlington, VA 22209

Dr. J. Richard Fisher
Assistant BMD Program Manager
U.S. Army
Strategic Defense Command
P. O. Box 15280
Arlington, VA 22215-0150

Mr. Robert Foord [2]
P.O. Box 1925
Washington, D.C. 20505

Director [2]
National Security Agency
Fort Meade, MD 20755
ATTN: Mr. Richard Foss, A05

Mr. Bert Fowler
Senior Vice President
The MITRE Corporation
P.O. Box 208
Bedford, MA 01730

Dr. Larry Gershwin
NIO for Strategic Programs
P.O. Box 1925
Washington, D.C. 20505

Dr. S. William Gouse, W300
Vice President and General
Manager
The MITRE Corporation
1820 Dolley Madison Blvd.
McLean, VA 22102

Dr. William Happer
559 Riverside Drive
Princeton, NJ 08540

Dr. Edward Harper [2]
SSBN, Security Director
OP-021T
The Pentagon, Room 4D534
Washington, D.C. 20350

Dr. Donald A. Hicks [2]
Under Secretary of Defense (R&E)
Designee
Office of the Secretary of
Defense
The Pentagon, Room 3E1006
Washington, D.C. 20301

Mr. R. Evan Hineman
Deputy Director for Science
& Technology
P.O. Box 1925
Washington, D.C 20505

Mr. Ben Hunter [2]
1917 Westmoreland Street
McLean, VA 22101

The MITRE Corporation [25]
1820 Dolley Madison Blvd.
McLean, VA 22102
ATTN: JASON Library, W002

Dr. Sherman Karp [3]
DARPA/STO
1400 Wilson Boulevard
Arlington, VA 22209

Mr. Ed Key
Vice President
The MITRE Corporation
P.O. Box 208
Bedford, MA 01730

Dr. George A. Keyworth
Director
Office of Science & Tech. Policy
Old Executive Office Building
17th & Pennsylvania, N.W.
Washington, D.C. 20500

MAJ GEN Donald L. Lamberson
Assistant Deputy Chief of Staff
(RD&A) HQ USAF/RD
Washington, D.C. 20330

Dr. Donald M. LeVine, W385 [3]
The MITRE Corporation
1820 Dolley Madison Blvd.
McLean, VA 22102

Mr. John McMahon
Dep. Dir. Cen. Intelligence
P.O. Box 1925
Washington, D.C. 20505

Mr. Charles Mandelbaum
Mail Stop ER-32/G-226 GTN
U.S. Department of Energy
Washington, D.C. 20545

Dr. Marvin Moss [2]
Technical Director
Office of Naval Research
800 N. Quincy Street
Arlington, VA 22217

Dr. Julian Nall [2]
P.O. Box 1925
Washington, D.C. 20505

Director
National Security Agency
Fort Meade, MD 20755
ATTN: Mr. Edward P. Neuburg
DDR-FANX 3

Prof. William A. Nierenberg
Scripps Institution of
Oceanography
University of California, S.D.
La Jolla, CA 92093

Dr. Robert Norwood [2]
Office of the Assistant Secretary
of the Army
(Research Development
& Acquisition)
The Pentagon
Room 2E673
Washington, D.C. 20310-0103

The MITRE Corporation
Records Resources
Mail Stop W971
McLean, VA 22102

Mr. Richard Reynolds
Director
DARPA/DSO
1400 Wilson Blvd.
Arlington, VA 22209

Mr. Alan J. Roberts
Vice President & General Manager
Washington C^3I Operations
The MITRE Corporation
1820 Dolley Madison Boulevard
McLean, VA 22102

Los Alamos Scientific Laboratory
ATTN: C. Paul Robinson
P.O. Box 1000
Los Alamos, NM 87545

Dr. Oscar S. Rothaus
106 Devon Road
Ithaca, NY 14850

Dr. Phil Selwyn [2]
Technical Director
Office of Naval Technology
800 N. Quincy Street
Arlington, VA 22217

Dr. Eugene Sevin [2]
Defense Nuclear Agency
6801 Telegraph Road
Room 244
Alexandria, VA 22310

Mr. Shen Shey
Special Assistant for
Directed Energy
DARPA
1400 Wilson Blvd.
Arlington, VA 22209

Dr. Joel A. Snow [2]
Senior Technical Advisor
Office of Energy Research
U.S. DOE, M.S. E084
Washington, D.C. 20585

COMO William Studeman
Director of Naval Intelligence
Office of Naval Intelligence
Navy Department
Washington, D.C. 20350

Mr. Alexander J. Tachmindji
Senior Vice President & General
Manager
The MITRE Corporation
P.O. Box 208
Bedford, MA 01730

Dr. Vigdor Teplitz
ACDA
320 21st Street, N.W.
Room 4484
Washington, D.C. 20451

Mr. Tony Tether
Director
DARPA/STO
1400 Wilson Blvd.
Arlington, VA 22209

Dr. Al Trivelpiece
Director, Office of Energy
Research, U.S. DOE
M.S. 6E084
Washington, D.C. 20585

LTCOL Simon Peter Worden
Strategic Defense Initiative
Organization
1717 H. Street
Room 416
Washington, D.C. 20301

Dr. Gerold Yonas [2]
Office of the Secretary of
Defense
Strategic Defense Initiatives
The Pentagon
Washington, DC 20301-7100

Mr. Leo Young
OUSDRE (R&AT)
The Pentagon, Room 3D1067
Washington, D.C. 20301

Mr. Charles A. Zraket
Executive Vice President
The MITRE Corporation
P.O. Box 208
Bedford, MA 01730

END

FILMED

12-85

DTIC