ADEPT — A HEURISTIC PROGRAM FOR PROVING THEOREMS OF GROUP THEORY

by

LEWIS MARK NORTON

S.B., Massachusetts Institute of Technology
(1962)

SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE
DEGREE OF DOCTOR OF
PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF
TECHNOLOGY

September, 1966

Signature of Author... Lewis M. Norton .....
Department of Mathematics, August 22, 1966

Certified by..... Marvin Minsky .....
Thesis Supervisor

Accepted by..... Norman Levitt .....
Chairman, Departmental Committee
on Graduate Students

ADEPT — A HEURISTIC PROGRAM

FOR PROVING THEOREMS OF GROUP THEORY

by

Lewis Mark Norton

Submitted to the Department of Mathematics on August 22, 1966 in partial fulfillment of the requirement for the degree of Doctor of Philosophy.

ABSTRACT

A computer program, named ADEPT (A Distinctly Empirical Prover of Theorems), has been written which proves theorems taken from the abstract theory of groups. Its organization is basically heuristic, incorporating many of the techniques of the human mathematician in a "natural" way. This program has proved almost 100 theorems, as well as serving as a vehicle for testing and evaluating special-purpose heuristics. A detailed description of the program is supplemented by accounts of its performance on a number of theorems, thus providing many insights into the particular problems inherent in the design of a procedure capable of proving a variety of theorems from this domain. Suggestions have been formulated for further efforts along these lines, and comparisons with related work previously reported in the literature have been made.

Thesis Supervisor: Marvin L. Minsky
Title: Professor of Electrical Engineering

ACKNOWLEDGMENTS

Work reported herein was supported by Project MAC, an M.I.T. research program sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract Number Nonr-4102(01). Reproduction in whole or in part is permitted for any purpose of the United States Government.

I wish to acknowledge and thank Professor Marvin Minsky, Dr. Seymour Papert, and Professor Manuel Blum for their criticism and guidance of the work reported herein. I also wish to thank Joel Moses for many constructive discussions, as well as for his preparation of a special LISP system which was used in this project. Appreciation must also be expressed to Project MAC for its assistance, including the computer time which made this project possible. Finally, I am indebted to my wife, Judie, for her efforts in the preparation of this report.

TABLE OF CONTENTS

Chapter	Title	Page
	Abstract	2
	Acknowledgments	3
I.	Introduction	6
II.	Description of ADEPT — I	
	Notation, Conventions, etc	13
	Figure 1 Example of a possible proof tree	22
III.	Description of ADEPT — II	
	Structure of the Program	25
	Diagram I Flow chart for ADEPT	26
	Figure 2 Proof tree for example	31
	Diagram II Flow chart for SCANW	35
	Diagram IIa Flow chart for SCANW (PROCESSING OF IMPLICATIONS)	36
	Diagram IIb Flow chart for SCANW (GENERATION OF LEMMAS)	37
	Diagram IIc Flow chart for SCANW (DETERMINATION OF SUBSTITUTIONS)	38
	Diagram IId Flow chart for SCANW (PERFORMANCE OF SUBSTITUTIONS)	39
	Diagram IIe Flow chart for SCANW (BOOKKEEPING)	40
	Diagram III Flow chart for PUTON1	45
	Diagram IV Flow chart for PUTON2	46
IV.	Special-Purpose Heuristics	
	The Dynamic Process of Using and Creating ADEPT	52
V.	Discussion of Selected Problems	64
	Problem V-A	65
	Figure 3 Proof tree for Problem V-A	66
	Problem V-B	70
	Figure 4 Actual output from ADEPT	71
	Problem V-C	74
	Problem V-D	75
	Problem V-E	77
	Problem V-F	79
	Problem V-G	81
VI.	Limitations of the Present Program	84
	Problem VI-A	86
	Problem VI-B	88
	Problem VI-C	89
	Problem VI-D	90
VII.	Extensions of ADEPT	94

VIII. Future Possibilities and Conclusions..... 103

Appendix I. Other Work in Theorem-Proving..... 119

Appendix II. Theorems Proved by ADEPT..... 131

Appendix III. Listing of the Program..... 138

Index to Listing..... 139

Bibliography..... 173

Index..... 176

Biography of the Author..... 178

CHAPTER I

INTRODUCTION

Since its earliest days, research on artificial intelligence has been concerned with the mechanization of theorem-proving. Investigators have approached the problem from two directions, combinatorial and heuristic. Programs have been constructed which operate in an essentially combinatorial manner on statements of the predicate calculus. Due to the work of Herbrand and Gentzen, it was known how to describe a procedure, completely mechanical in nature, that would be able to prove every "provable" theorem, sooner or later. Here the word "provable" is used in its technical sense, and the reader may refer to any book on symbolic logic. A proof procedure such as Herbrand's is known as a complete proof procedure, and in the early days of digital computers it was not universally appreciated that even the large speed and memory of the computer would not suffice to make implementation of a complete proof procedure practical for problems of respectable complexity. However, due to results obtained with early codings of Herbrand's procedure, it was soon discovered that the inefficiency of these algorithms quickly exhausted the resources of the largest and fastest computers. Much work has subsequently gone into refinements and modifications of complete proof procedures, and considerable improvements have been made. Some programs of this nature will be discussed in an appendix.

The other approach to proving theorems by computer has been the heuristic one. Projects in this class are not to be confused with those programs, essentially combinatorial in nature, which have been modified by the inclusion of heuristics. Rather, the basic structure of the algorithm must be heuristic in nature. The earliest such attempt, and an important one, was the Logical

Theorist of Newell, Shaw, and Simon.[14] Further work by these authors[15] was joined by the so-called "advice-taker" approach originated by McCarthy.[7] Many of these programs are in reality general reasoning algorithms, and their use as theorem-provers was done merely by way of an example. The heuristic approach holds out the enticing possibility of capturing the essence of human procedures which enable difficult problems to be handled with a reasonable amount of effort. The price to be paid is that of expressing, in the form of a computer program, most of the large number of methods that people use, often without realizing that they are doing so, while solving problems.

The work to be discussed here belongs in the second category. Its development, however, has been of a different nature than that of most previous heuristic problem solvers. In this case, there has been no attempt to develop a general problem solving mechanism. Instead, attention has been focussed on the abstract theory of groups[22] (not necessarily finite), and in particular on the question of an algorithm uniquely designed to handle theorems of this particular area of pure mathematics. As is related at the end of this report, the program actually constructed can be easily adapted to "advice-taker" problems in other areas — a fact hardly surprising in view of the reasoning capabilities needed to solve theorems in any branch of mathematics. But the creation of the program was done with only group theory in mind.

In fact, the program was created only as a step toward a computer procedure for solving difficult problems in the theory of groups. Just as it was necessary to actually program Herbrand's procedure in order fully to appreciate the difficulties which arise in using this method with a digital computer, so it is necessary to have a program capable of handling a fair number of group theory theorems of slight to medium difficulty in order fully to appreciate the difficulties inherent in coping with the theory of groups using a digital computer. Existing programs had proved a few elementary results in group theory, and more are being done as time goes on, but no program reported in the literature has proved more than a handful of theorems from this subject, and in particular, very little work in group theory has

been done by programs of a basically heuristic nature.

Most heuristic proof procedures, including the one to be discussed in this report, employ a main routine which works backward. That is, it begins with the statement which is to be proved, and constructs a chain of statements in the hope of reducing the original statement to one which is known or can be established. The chain is so constructed that if such a statement is found, the desired conclusion is proved or partially proved. In other words, at any point, the chain is part of a "proof tree" leading, by means of valid inferences, to the desired conclusion, though no part of it will be a true proof tree until some verifiable branches are generated. There is, usually, no guarantee that the chain will be turned into a true proof. The alternative is a forward approach, which consists of deriving statements from the hypotheses and axioms. In this approach, all chains are true proofs; the problem is that most statements proved are of no relevance to the desired result. In (14), Newell, Shaw, and Simon argue for the "working backward" approach. They liken it to the process a needle would use to find its way out of a haystack, as opposed to a search for a needle in a haystack.

In addition to the basic structure of a program (that is, the organization of its main routine), there are the special-purpose subroutines which enable it to perform capably in a complex problem area. These subroutines may incorporate individual heuristics; what is more, if the main routine is properly organized, they can be altered in various ways, thus making it possible to perform experiments which will help determine more successful methods for dealing with the problem area in question. This is certainly true with heuristic theorem-provers. In developing the main routine, one must decide such questions as "in which direction is a proof to be developed?" But the task of establishing theorems can only be done with the additional aid of many special-purpose techniques, each of which must be incorporated somehow into the basic program.

One of the advantages of the heuristic approach is that various techniques used by human problem-solvers can be directly incorporated into

the program in a natural way. This was definitely a major consideration of the present project. Admittedly, it is not true that people organize their problem-solving efforts exclusively in the "working backward" manner employed by this particular heuristic program. However, it is fair to say that a large amount of theorem-proving is done in that fashion, and certainly a proof proceeding in that manner is "natural" in appearance. Similarly, it is easy to attach various subroutines to a main routine organized to proceed backward.

With the organization outlined above, a computer program was written, designed to produce efficient and natural-appearing proofs for theorems of group theory. The program is known as ADEPT, which stands for "A Distinctly Empirical Prover of Theorems". As its name indicates, ADEPT is extremely ad hoc, which should not be surprising when one considers how non-compact and poorly formalized are the methods that people use. Thus ADEPT does not have a neat, transparent algorithm, and any reader who feels that all mechanical theorem-provers should be basically tidy will be disappointed in the present work from that viewpoint. However, compensating for the complexity of the program is the ease with which ADEPT may be augmented with direct analogues of various shortcuts which people successfully use, and in this respect rapid development took place in this project. A notation was developed which was simple to use, incorporating English terms in a "Cambridge Polish" format.(8) This helped obtain flexibility for experimenting with special-purpose heuristics, and ADEPT became a useful tool for investigating the problem area of group-theoretic theorems.

A large percentage of the special-purpose heuristics used with ADEPT restrict the growth of the proof tree. Ideally, no irrelevant steps should be allowed by such heuristics, and no step necessary to a correct and efficent proof should be prevented from occurring. In practice, of course, the heuristics are imperfect and fail to some degree in both respects. What seems to be needed is for the program to understand where it is in a proof; to be aware of what it is doing, and why. It is of no interest here to argue over whether or not a program can ever "be aware"; the effect is what is of

interest. A program which generates efficient proofs for a large number of theorems is the goal, and if it is achieved one might as well say that the program is "aware". Certainly the benefits of awareness will have been obtained.

A great amount of effort went into achieving such a property in ADEPT. In fact, a strong requirement was adopted from the beginning. One algorithm was to suffice for all theorems, yet be efficient in all cases. ADEPT never "backtracks". If for some reason it fails to prove a theorem, it does not remove some or all heuristic restrictions, though this could be done rather easily. Until the program is improved, enabling it to handle that theorem without any serious loss of efficiency in other proofs that it has successfully produced, it will just be unable to prove that theorem, efficiently or inefficiently. This requirement may seem harsh, for people sometimes quit and start over in a different way, but it must be remembered that ADEPT is not doing advanced problems. Thus it is reasonable to experiment and find out just how far this "one-pass" approach can be carried.

Closly related to the idea of "awareness" is the question of ascertaining when progress is being made in a proof. There is no simple way to do this — no metric, no compact test. As will be seen, much of the effort of developing a program which appears to understand what it is doing went into an analysis of how to recognize those steps which constitute real advancement toward the desired goal.

The construction of a theorem-prover using this philosophy did not occur all at once. An "evolutionary" process was involved, as the fine points of the problem became evident. Thus the earlier versions of ADEPT served as vehicles for understanding the task of theorem-proving better, and for suggesting new heuristics to be incorporated into the program. This report tries to make this developmental aspect of the project clear.

The presentation of both the development of ADEPT and its latest version is supplied with numerous, detailed examples. ADEPT has been used to solve many theorems, and each has provided new insights into the needs of any

algorithm which must cope with many diverse, individual theorems. Examples are given to illuminate the reasons for various additions which have been made to ADEPT. Others illustrate the more interesting abilities of ADEPT to prove problems efficiently. Still others point out the deficiencies which still remain. The totality represents a long program of experimentation, and underlines the worth of a working system which can be readily used to explore a problem area in detail.

The theorems presented to ADEPT are all familiar to a student of group theory. For the most part, they are given in this report using a common mathematical notation. Those terms which represent concepts which must be defined in order to specify the theorem will be underlined when used in the statement of a theorem. From a syntactic point of view, of course, the precise statement of the hypotheses, conclusion and relevant definitions determine the problem to be proved. A slight variation in these statements results in a completely different problem. However, such a variation may not be a different problem from a semantic viewpoint; i.e., its interpretation in group theory may be equivalent. It is also possible that a variant of a problem may have a different (e.g., broader) interpretation, yet result in the same operations by the program. Experiments have been carried out involving alternate versions of a number of problems, and indeed the notation used by ADEPT allows flexibility in arranging such experiments. In general, the view is adopted that if ADEPT can prove some but not all of two or more semantically equivalent but syntactically different versions of a theorem, ADEPT "can prove the theorem".

With this introduction, it is time to begin the actual detailed report. Chapter II introduces the notation (i.e., syntax) used by the program, and presents other details necessary to understand the description of the operation of the program itself, which is given in Chapter III, accompanied by numerous flow charts. The fourth chapter specifically discusses ADEPT's special-purpose heuristics — their development, purpose, and operation. Then, accounts of detailed problems commence, and it is felt that the number

and length of the problem descriptions are extremely valuable for an understanding of this project. Theorems successfully proved by ADEPT are the main content of Chapter V, while Chapter VI concentrates on those theorems which, for any of a number of reasons, the program cannot handle. Chapter VII is a description, with examples, of extensions to the ADEPT program which have actually been carried out, though in some cases rather crudely. Then Chapter VIII suggests possible alterations, additions, and directions for future work which have not been explored to date. An appendix reviews the field of mechanical problem-solving in the light of the present effort, covering both heuristic and combinatorial programs which have been reported in the literature. A bibliography and index supplement the text.

CHAPTER II

DESCRIPTION OF ADEPT — I

NOTATION, CONVENTIONS, ETC.

ADEPT has been programmed in the language LISP 1.5,[8] using a special version of the system created by Joel Moses, a variant of the system currently in use at MIT's Project MAC.[4] The special version provides an increase of a thousand words (or 14%) of binary program space by eliminating those features of the standard LISP package which are not used by the ADEPT program. As much of ADEPT as is possible is compiled. No new features were added to the LISP system.

Since ADEPT is programmed in LISP 1.5, it is advantageous to have statements of its own language appear as lists to LISP. As will be seen in this chapter, the notation adopted for representing mathematical statements possesses this property. In particular, the basic (or atomic) entities of this notation form a subset of LISP's atomic symbols. For ADEPT, a symbol is a string of alphanumerics, the first of which is a letter or an asterisk. Symbols are delimited by blanks or parentheses.

To describe the language used by ADEPT, one must first designate which symbols are used for variables and constants. Three types or sorts of variables and constants are provided for in the basic system, though more types could be added in the future, as mentioned in Chapter VII. The same symbols are used for both the variables and constants of a given type, and a flag associated with each symbol indicates which use is being made of it at a given time. Single letters (A through M, excluding F because of its special role in the LISP language) are used to designate sets, the symbols F1 through F9 are used for functions, and the symbols A1 through A12 represent variables which may be members of sets. In addition, the program is provided with a facility

to define new variables or constants as needed by internally generating new, unique symbols.

By choosing the variable types as above, a two-level hierarchy of membership is fixed. Clearly ambiguity is possible with such a system, as, for instance, with a coset, which while it is a set with its own members, also can be itself a member of a larger set, the factor group. The burden of deciding whether such an object must be considered to be a member or a set in a particular problem rests with the user, who must use one type of symbol or the other when he states the problem. This limitation caused no difficulties while experimenting with ADEPT; i.e., no problem required three or more explicit levels of inclusion.

The language contains connectives or propositional terms, and the following symbols are recognized as such: AND, OR, NOT, IMPLIES, IMPLIES2. IMPLIES2 is the notation for the biconditional (<==>). In addition, the language presently contains three logical constants, EQUAL2, EQUAL, and FEQUAL, all of which are used to represent equalities. EQUAL2 is the basic symbol, and the "2" emphasizes the symmetric nature of equality. EQUAL and FEQUAL are present to allow the user easy implementation of some heuristic options. The use of EQUAL in a statement will restrict ADEPT to substitutions in only one direction, as will be seen in Chapter III. This makes it possible for the user, by his choice of logical constant for stating an equality, to provide that an axiom such as $a_1 a_1^{-1} = e$ allow only simplifications, thus preventing expansions. An equality to be proved, stated using FEQUAL, will be processed differently by ADEPT from one stated using EQUAL2 or EQUAL, and this feature will also be described in the next chapter. Experience has shown that the flexibility provided by these options is of tremendous value. (When mathematical notation is used in this report to describe a proof, the symbol = will ordinarily be used for equality. However, if confusion is present or if it is important to specify the exact form of a statement in ADEPT's language, mathematical notation will be augmented by the symbols $=_2$, $=_f$, and $\leftarrow$, corresponding to EQUAL2, FEQUAL, and EQUAL respectively, and the notation for EQUAL is

intended to emphasize that the second argument can be substituted for the first, but not vice versa.)

The symbol EXISTS is used as an existential quantifier, but there is no explicit universal quantifier. This should not be surprising, for most uses of the universal quantifier in group theory are implicit. Any statement with variables not bound by an existential quantifier is interpreted as being universally quantified over those variables. As for the existential quantifier, it will be seen in the next chapter that ADEPT does not use a uniform procedure to handle it. Instead, special cases are provided for separately, and new subroutines can be created as the need for treating more cases arises.

The name syntactic constant will be used to group together connectives, logical constants, and quantifiers. The operations performed by ADEPT upon encountering a syntactic constant will be discussed in Chapter III.

Any symbol which is not a variable/constant or a syntactic constant is a term. Terms can be identified by entries in tables of necessary and sufficient conditions. When the operation of ADEPT is considered, it will be explained why there are in fact only two tables — one of sufficient conditions and one with definitions (necessary and sufficient conditions). There are two terms which are given null entries in these tables — MEMBER and *PROD — and thus may be considered to be primitives.

Terms may start either with an asterisk or with a letter, and it will be seen that asterisked and non-asterisked terms have different roles in the syntax of the language. Consequences of the use of the two kinds of terms will appear throughout this report.

It is now time to see how the various symbols are combined into statements. First, it is necessary to describe an intermediate notion which is essentially a generalization of the concepts of variables and constants, and which also is subdivided into types. Objects are defined (recursively) as follows:

i) a set variable or constant is a set object (e.g., A);

ii) a member variable or constant is a member object (e.g., A1);

iii) a list of three elements, the first of which is a function variable or constant and the second and third of which are set objects, is a function object (e.g., (F1 A B));

iv) a list of four elements, the first three of which are as in iii) and the last of which is a member object, is a member object (e.g., (F1 A B A1));

v) a list of one or more elements, the first of which is an asterisked term and the rest of which are objects of any type, is an object of type to be determined by the user's interpretation (e.g., (*CENTER G)).

In iii), iv), and v), the compound objects, the elements of the list excluding the first are arguments of the first element, and the list is said to be headed by its first element. All lists conform to the syntax of the programming language LISP 1.5.

Some comments on the concept of object are in order. The definition by iii) of function objects and the closely related definition by iv) of certain member objects are illuminated by the intended interpretation. The second and third elements of these lists are to be understood as the domain and range, respectively, of the function. The fourth element of the list, if present, is the argument of the function in the mathematical sense. Thus (F1 (*INTERSECTION H K) K) might denote the canonical map from $H \cap K$ into K, and (F1 (*INTERSECTION H K) K A1) would then denote some point of K, namely the image of A1 under the map in question. Note that function variables or constants are not allowed to "stand alone" as function objects. All functions have a domain and range, and the language, unlike mathematical notation, requires that this must be made explicit. Incidentally, the syntax allows objects that have no interpretation. For instance, the object (F1 A B A1) is meaningless unless A1 is contained in A.

The intended interpretation also supplements the definition of objects given by v). Here the user, in formalizing a definition or sufficient condition for an asterisked term, will determine what arguments are needed, and

therefore their type. (*IDENTITY G) is a possible example of a member object with one argument, a set object. (*INTERSECTION H K) was used above as a set object with two set objects as arguments. Thus the asterisked term in this kind of object syntactically is a function.

Objects are combined with the other symbols into statements or formulas according to the following recursive definition (all statements will be lists to LISP 1.5):

i) a list of two or more members, the first of which is a non-asterisked term and the rest of which are objects of any type, is a statement (e.g., (SUBGROUP H G));

ii) a list of three elements, the first of which is EQUAL, EQUAL2 or FEQUAL and the other two of which are member objects, is a statement (e.g., (EQUAL2 A1 (*IDENTITY G)));

iii) a list of two elements, the first of which is NOT and the second of which is a statement, is a statement;

iv) a list of three elements, the first of which is AND, OR, IMPLIES, or IMPLIES2 and the other two of which are statements, is a statement;

v) a list of three elements, the first of which is EXISTS, the second of which is a variable, and the third of which is a statement, is a statement.

Clearly, meaningless statements are possible; i.e., ones with no interpretation. However, the intended use of the various forms of statements should be clear, except for those defined by i). There is, of course, a parallel between such statements, headed by a non-asterisked term, and the objects headed by an asterisked term. The arguments in each are determined by the user's use of the term. However, a statement headed by a term is interpreted as a statement about its first argument. Thus (CENTER C G) says that C is the center of G, and (HOMOMORPHISM (F1 G H)) says that a certain map is a homomorphism. Non-asterisked terms, then, serve as predicates, with a restricting convention on the order of the arguments.

Any statement which contains an object headed by an asterisked term can be translated into one that does not. For instance, the statement (INTERSECTION I (*CENTER G) H), which says that I is the intersection of the center of G and some set H, could be restated as (AND (CENTER C G) (INTERSECTION I C H)). This is a direct parallel to the practice of introducing variables in a proof stated in mathematical notation; e.g., "Let C be the center of G, and consider the intersection I of C and H. Then...." Though such a translation is always possible, it is not always desirable. For instance, a user would not want to eliminate an instance of (*INVERSE A1 G) any more than a mathematician would want to say "let y be the inverse of x" instead of simply using "x^{-1}". (The symbol G denotes the set on which composition, and hence the inverse, is defined.)

A converse translation, eliminating non-asterisked terms in favor of asterisked terms, is not possible. How, for instance, could "$a_1 \in G \ \& \ a_2 \in G$" be expressed, since both (MEMBER A1 G) and (MEMBER A2 G) could only be translated into (*MEMBER G), thus losing information? An object such as (*MEMBER G) would be a variable, syntactically. To simplify construction of ADEPT's matching routines, a convention has been adopted providing that <u>all objects headed by an asterisked term must be used as constants</u>. Thus (*MEMBER G) would be excluded (except in the unlikely case that G has but one member), while (*CENTER G) is an object which could appear in a statement. Translation from statements headed by non-asterisked terms is therefore limited to cases where the term <u>uniquely specifies</u> its first argument.

While use of asterisked terms is natural in some cases, such as the above-mentioned example of inverses, at other times it is not obvious whether or not such terms should be used. A number of examples will be given in later chapters illustrating the effects of notation on specific problems. It suffices to say here that ADEPT's language allows a degree of flexibility in this regard which sometimes can be exploited to advantage. The program itself has no facilities for translating from the use of one type of term to the use of the other.

All proofs constructed by ADEPT are lists of statements, developed by operations on statements, so sufficient information on notation has been presented to allow discussion of a proof. However, for purposes of programming, many statements have associated property lists, which are inserted into a list which is a true statement, preceding the first element. This is the standard form of a table entry. In LISP terminology, CAR of a table entry is the property list, and CDR of a table entry is a statement. For example, statements to be entered in the tables of conditions and definitions are accompanied by a property list consisting of the term being specified, followed by variables of appropriate type serving as dummy arguments. An example will clarify: a sufficient condition for subgroup (closure under products and inverses) is given by ((SUBGROUP A B) IMPLIES (AND (MEMBER A1 A) (MEMBER A2 A)) (AND (MEMBER (*PROD A1 A2 B) A) (MEMBER (*INVERSE A1 B) A))). Now if it were desired to expand the statement (SUBGROUP H G) using this condition, the desired substitution instance of the statement given in the table entry could be obtained easily with the aid of the result of matching the property list with the statement to be expanded. Note how the property list of a definition or condition specifies the number and type of arguments of a term. The primitives MEMBER and *PROD have their arguments similarly specified by null definitions — ((MEMBER A1 A)) and ((*PROD A1 A2 A)) respectively (where the third argument of *PROD is the set on which the composition is defined). Needless to say, consistency in the use of arguments must be followed in creating the tables of conditions and definitions.

To complete the groundwork for the discussion in Chapter III of the operation of the ADEPT program, it is desirable to discuss the way in which a proposed theorem is inputted to the program, and the output that is obtained from it. Instrumental to this discussion is the fact that in the course of working on a proof, ADEPT develops two lists, or tables. Table I is a list of known statements — either hypotheses, axioms, or certain intermediate results; table II is the proof tree proper.

A problem is presented to ADEPT in three statements, each of which may be

a conjunction. The first two are hypotheses, either (or both) of which may be vacuous. The first of these will have all of its variable/constant symbols fixed (i.e. flagged) as constants; any additional variable/constant symbols in the second will not be flagged. Both statements are then "sent" to a subroutine called PUTON1 (described in the next chapter) for inclusion on table I. The third statement of input is the desired conclusion. All of its variable/constant symbols will be fixed as constants except those appearing in (sub)statements headed by the connectives IMPLIES or IMPLIES2. The conclusion will be put on table II by the subroutine PUTON2 (also to be described in Chapter III). No syntax check is performed on any statements inputted or internally formed to insure that they are legally formed, so the user must exercise care.

In this way the determination of whether variable/constant symbols are used as constants or variables is handled during the inputting of a problem to ADEPT. Any such symbol not flagged as a constant is treated as a variable. In the course of a proof, additional symbols may be generated, but the purpose for the generation will specify their use. A restriction on the user in this regard is actually the only unfavorable consequence observed due to the absence of an explicit universal quantifier in the language. This is the fact that any symbol intended to be a variable in an implication which is part of a conclusion must not have been previously fixed as a constant due to its use in the first hypothesis. For example, if (LCOSET A A1 B C) were the first hypothesis (A is the left coset of B given by A1 where composition is defined on the set C — e.g., B is a subgroup of C and A is a_1B) then (IMPLIES (MEMBER A1 A) (MEMBER A1 C)) would not be a way of stating the conclusion that A is a subset of C. The intent, having the A1 of the conclusion be a universally quantified variable, is thwarted by the previous treatment of A1 in the hypothesis as a constant. Thus the conclusion presented as above would result in a proof that just A1, the particular element specifying the coset, is a member of C. (Note that the statement (IMPLIES (MEMBER A1 (*LCOSET A1 B C)) (MEMBER A1 C)) as a conclusion would also have the restricted interpretation, but this statement

could not have any other interpretation anyway!) Use of A2 in the conclusion would remedy the problem, which is an uncommon one, since other methods of stating conclusions avoiding implications are usually not only possible but more natural, as in this case, the statement (SUBSET A C). The user must also be careful when using implications in conclusions to insert hypotheses which will fix as constants any variable/constant symbols which must be so treated in the implication. This conceivably could necessitate dummy hypotheses. Alternatively, the whole problem (which seldom is encountered) could have been avoided by requiring the user to accompany the three input statements with a list of which variable/constant symbols are to be treated as constants.

The manner in which the proof tree is developed is the subject of the next chapter, but a few remarks will now be made concerning the structure of table II. Basically, it is an embodiment of the "working backward" approach to theorem-proving.[14] As indicated above, the desired conclusion becomes the first node, or head, of the proof tree. Loosely speaking, the proof procedure may be said to be an attempt to reduce the desired result to a statement which is already established. For this reason, branches of the tree are called reductions. A reduction of a node is, with one exception, a statement which, if verified, would suffice to verify that node. A node, of course, may have any number of reductions. The exception referred to occurs at a node which is a statement headed by the connective AND. Such a node is immediately subdivided and therefore has two and only two reductions, its conjuncts, each of which must be verified in order to verify the conjunction. The notation of Slagle [23] will be used in diagrams of proof trees; reductions are indicated by lines descending from the parent node, and reductions which must both be proved in order to verify the parent node are linked by an arc (Figure 1).

Statements on both tables I and II have associated property lists. The table I entries are accompanied by a line number and a numerical indicator of the logical class to which the line belongs, a concept which will be developed in Chapter III. The property list of a table II entry is more complex. Its composition is as follows:

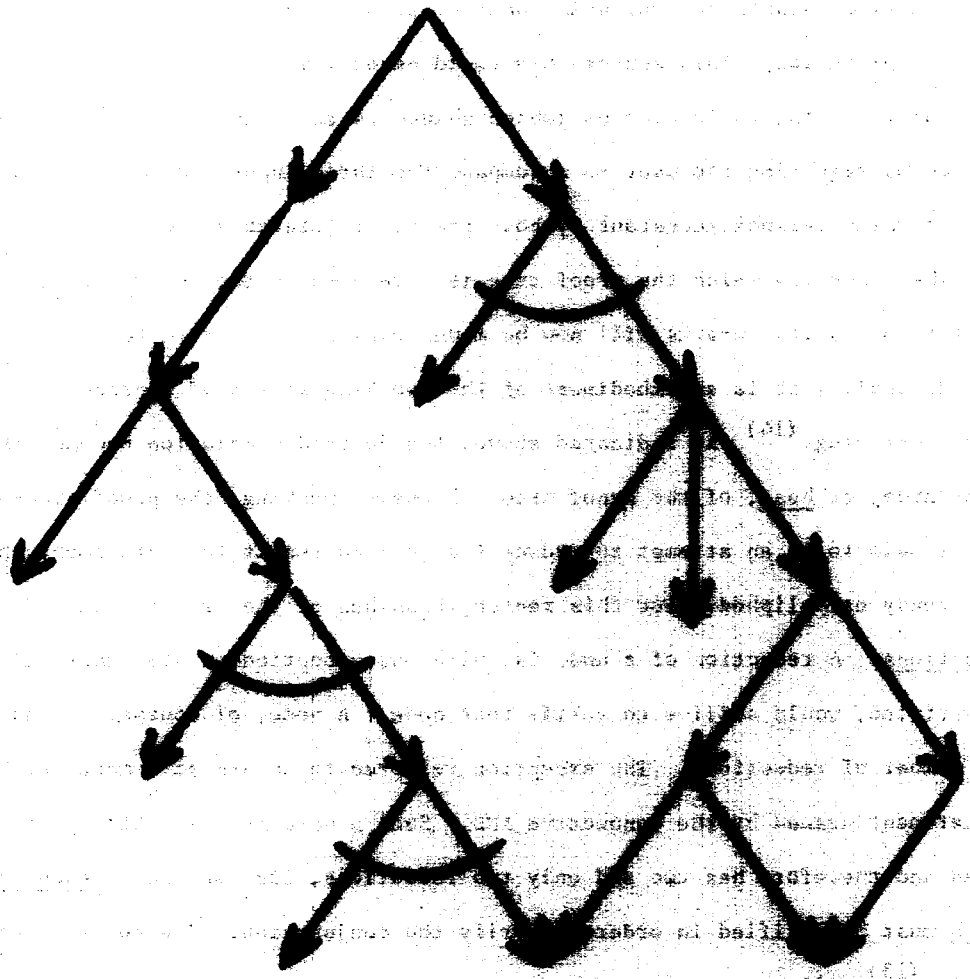


Figure 1

Example of a possible proof tree

i) an indicator of the line's status;

ii) a list of the line numbers of the line's immediate predecessors;

iii) a list of the line numbers of the line's reductions;

iv) the line's number;

v) the line number assigned or that will be assigned to the first entry on table I which has not been checked against the line;

and an optional element present only on the property lists of lines which are heads of subordinate trees begun in the course of a proof:

vi) the line number of the table I entry which was used in the creation of the line.

The use of these elements will become clearer in the succeeding chapters.

The set of possible statuses for a proof tree line is as follows:

i) VER - verified;

ii) REL - unverified and relevant;

iii) RELl - unverified, relevant, but deemed to be partially processed;

iv) IRR - known to be irrelevant;

v) SIM - unverified, but simplified (or subdivided, in the case of a conjunction) and therefore to be skipped;

vi) ST - on a subordinate tree and unverified;

vii) VERST - on a subordinate tree and verified;

viii) PNT - that line which is currently under closest examination by ADEPT. (PNT is an acronym for "pointer".)

In discussing the formal language and proof structure used in the ADEPT project, no mention has been made of rules of inference. This is because they are not treated formally, but in an ad hoc manner, as will be seen in the detailed description of ADEPT's operation.

ADEPT has been implemented on CTSS (Compatible Time Sharing System)[2] at MIT's Project MAC. Therefore, communication with the program from a

teletypewriter console is possible. Stating a problem is done by typing in the three statements described previously, and conditions and definitions will be requested if not found in the tables. An answer of "NO" will be recorded as a null condition or definition so that terms may be undefined without repetitive demands from the system for the definition. The output consists of the final status of tables I and II. (Sometimes information on property lists is destroyed when table II lines are verified, so the final output may appear to have lines with erroneous property lists.) One observes from the output whether or not a theorem has been proved simply by noting whether or not the first entry of table II is of status VER.

CHAPTER III

DESCRIPTION OF ADEPT — II

STRUCTURE OF THE PROGRAM

As was pointed out in the last chapter, ADEPT uses a "working backward" method of proof. This basic organization is outlined in somewhat more detail in Diagram I. Examination of that flow chart will reveal that the majority of the work of proving a theorem must take place during the steps labelled "explore consequences...". Also it is obvious that much must be explained about the use of the innocent word "progress".

Some of the occasions when entries are made to tables I and II are evident from Diagram I. Many others are found in the "explore consequences" phases of the program. The rest of the basic structure of ADEPT lies in the routines which add lines to one of these tables; i.e., add a property list to a statement and do any necessary processing which must accompany the placement of the new table entry. Of course, there are also matching subroutines, and a "tree-pruning" subroutine which is called whenever a table II entry is verified.

A few remarks regarding the tables of sufficient conditions and definitions are now in order. First of all, it is important not to confuse these tables with table I, which is the list of "known" information being used in any one proof. The totality of information included as definitions, etc., is never dealt with in a proof; only those instances of the entries of these tables which are put on tables I or II are processed in any detail.

Instances of sufficient conditions are used in only one way. As shown in Diagram I, if the line currently of status PNT is a statement headed by a term, then if there is a sufficient condition for that term in the table, the appropriate instance of that condition will be added to table II as a reduction

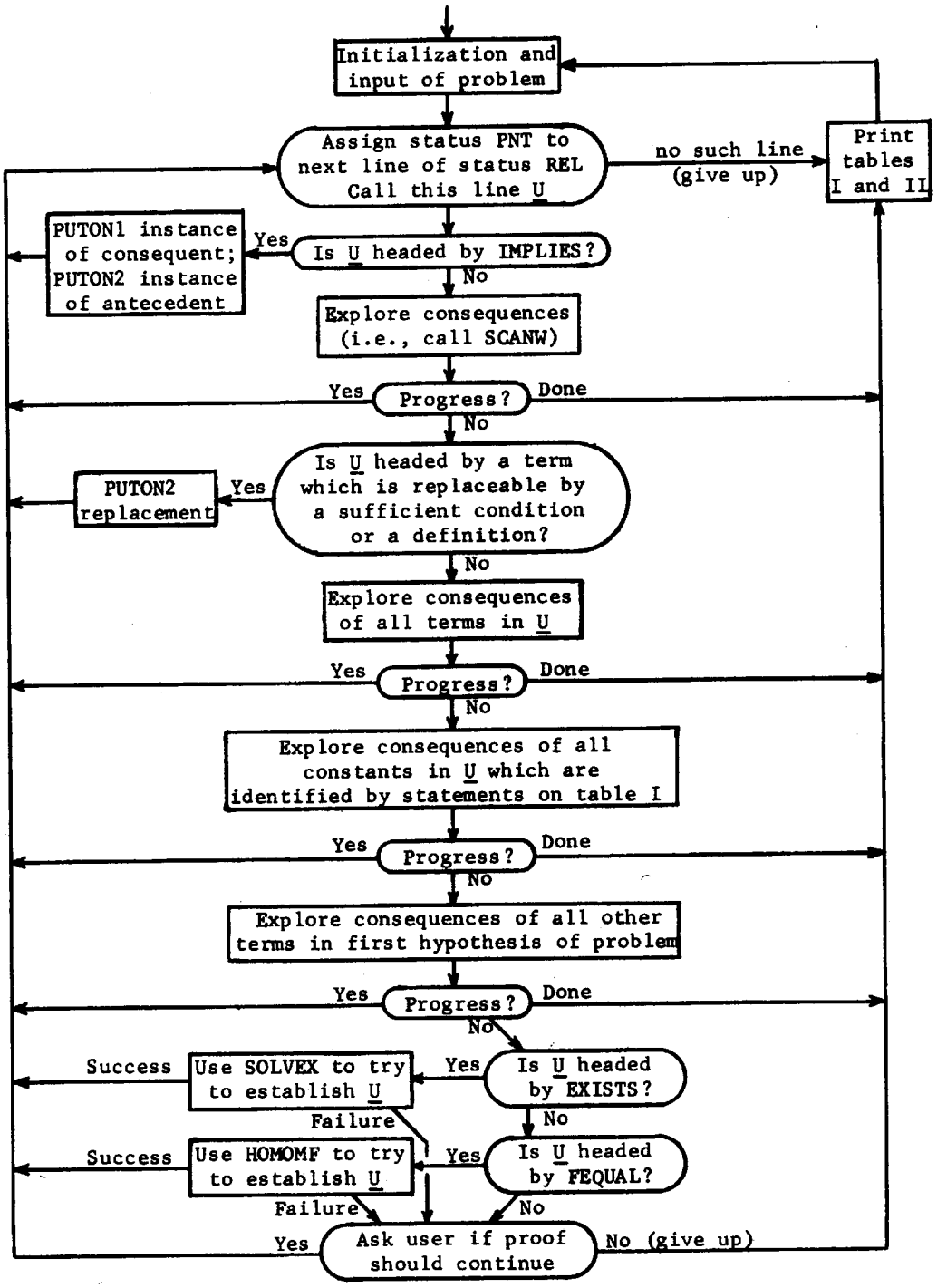


Diagram I

Flow chart for ADEPT

of the line of status PNT. In the event that a sufficient condition is not available, the table of definitions will be consulted for a possible reduction. Instances of definitions can also be placed on table I, and this occurs at three steps of Diagram I which are labelled with the phrase "explore consequences", augmented by "of...". In these cases, a call for definition instances is being made, either for definitions of terms appearing in a proof tree line of status PNT, terms in table I statements identifying constants in the line of status PNT (e.g., (SUBGROUP H G) identifies H), or terms appearing in the first hypothesis of the problem.

The method of proof utilized by ADEPT makes it natural to have a separate table of sufficient conditions, supplying a source of material for possible reductions. Definitions are indispensible for many reasons, and it would be poor practice to force a check of both a sufficient condition table and a necessary condition table in order to obtain them, hence ADEPT has a table of definitions. As for necessary conditions per se, they could be used for the same reasons that definitions are used in connection with adding statements to table I, but a need for such a feature has not been observed with ADEPT. The inclusion of such a table might be a future extension. Until such time, necessary conditions for terms may be included in the definition table if the user is sure for some reason that no attempt will be made by the program to add to table II an instance of the definition of that term.

At present, provision is made for only one sufficient condition or definition per term. Of course, this is hardly surprising in the case of definitions. However, even in the case of conditions, there is really no restriction because of the possibility of using the connective OR. This does lead to the equal consideration of all disjuncts, at least at present, and for harder problems requiring selection among multiple conditions, more sophisticated procedures will have to be implemented in order to overcome inefficiency.

In Diagram I, "initialization and input of problem" results in the placement of the two hypotheses on table I and the conclusion on table II. The conclusion, or more precisely, all conjuncts of the conclusion which are not

themselves conjunctions, will be assigned status REL. The main loop is then entered, and one can think of a pointer being positioned on the first table II line of status REL. Note that this assignment of status PNT is made only to lines of status REL; thus, for instance, lines on subordinate trees can never undergo the special treatment afforded to such lines.

If the current line of status PNT is an implication, an instance of it is created by generating a new symbol for each distinct variable (not constant) of the line, declaring these new symbols to be constants, and performing the indicated substitution (thus achieving a process of universal specification). The antecedent of the resulting implication is then added to table I and the consequent becomes a reduction (of status REL) of the original implication. (In general, reductions are tentatively assigned the status of their predecessor. See, however, Diagram IV.) This terminates consideration of the implication.

If the line of status PNT is not an implication, ADEPT at this point calls its main subroutine, which performs a scan, deriving reductions of proof tree lines which can be inferred from the current set of entries on table I. This derivation can be done in various ways, with various allowed inferences, and will be discussed in detail shortly. This is what is meant by "explore consequences". In some subsequent steps of Diagram I, it is done after adding appropriate definition instances, if any, to table I.

Treatment of special cases by means of the subroutines SOLVEX and HOMOMF, mentioned near the bottom of Diagram I, will be discussed at the end of this chapter. As for the communication to the user regarding continuation of a proof, clearly this is just a frill made possible by the on-line terminals of the MAC time-shared system. This point on the flow chart is almost never reached in theorems which ADEPT is able to prove.

Diagram I reveals to some extent the heuristic nature of ADEPT, at least on one level. Of course, proceeding back from the conclusion is itself a heuristic for proving theorems. In addition, existence of such entities as FEQUAL allows heuristic possibilities. But a central and most difficult goal of this

project was to introduce, presumably by use of heuristics, an "awareness" into the program. It seems obvious that a successful heuristic theorem-prover must perform as though it "understood" where it was in a proof. A student, of course, has such an ability, and to what degree it can be inserted in a program will be reflected in the order of development of a proof and in the ratio of necessary to irrelevant lines in the proofs produced.

The means used to attempt to achieve this ability in ADEPT include the use of the status PNT, and the accompanying design and application of criteria for "progress", which determine when the pointer is to be advanced. Through use of status PNT, instances of definitions can be introduced as they become relevant. Of course, this feature can be of help only until all the instances which will be used in a proof have been produced. In addition, if the point in the flow chart where terms of the first hypothesis are considered is reached, experience shows that most of the instances will be obtained all at once. However, the first stages of a proof often proceed more efficiently because of this "selectivity" feature. Alternative methods for achieving similar results are discussed in Chapter VIII. As for detailed discussion of particular heuristics, this is the subject matter of the next chapter.

The flow of a proof can now be illustrated, and the example will also serve to illuminate the content of Chapter II. The proof will be discussed in both ADEPT's language and mathematical notation. Rather than supply appropriate definitions to the tables, it will be assumed that the tables are empty (except for null definitions of *PROD and MEMBER, and a null sufficient condition for MEMBER) and that ADEPT asks the user for information at the appropriate times.

Theorem: If K is the kernel of a homomorphism f_1 mapping G into H, then K is a submonoid of G.

This is presented to ADEPT as follows:

```
(AND (KERNEL K (F1 G H)) (HOMOMORPHISM (F1 G H)))
NIL
(SUBMONOID K G).
```

(In what now occurs, it will be assumed that an early version of ADEPT is being used, unaugmented by a number of heuristics yet to be described.)

Table I now is as follows (since conjunctions and biconditionals are immediately subdivided by PUTON1):

((1 1) KERNEL K (F1 G H))

((2 2) HOMOMORPHISM (F1 G H)),

K, F1, G, and H are fixed as constants, and ((REL (HEAD) (NONE) 1 1) SUBMONOID K G) goes on table II. This line is changed from status REL to status PNT, and the preliminary scan is fruitless. ADEPT now desires a sufficient condition for submonoid, and the user responds ((SUBMONOID A B) IMPLIES (AND (MEMBER A1 A) (MEMBER A2 A)) (MEMBER (*PROD A1 A2 B) A)); i.e., A is a submonoid of B if $a_1 \in A$ & $a_2 \in A$ ==> $a_1 a_2 \in A$ (i.e., A is closed under composition). This results in the instance $a_1 \in K$ & $a_2 \in K$ ==> $a_1 a_2 \in K$ being added to table II as a reduction: ((REL (1) (NONE) 2 1) IMPLIES (AND (MEMBER A1 K) (MEMBER A2 K)) (MEMBER (*PROD A1 A2 G) K)). This causes the property list of line (1) (which had already been altered by the fruitless scan) to be changed to (REL (HEAD) (2 NONE) 1 3), as line (2) is assigned status PNT. Property lists will henceforth be ignored in this discussion, except that entries will be referred to by their line numbers.

The implication is immediately split, and, assuming the symbols generated are K1 and K2, the statement (AND (MEMBER K1 K) (MEMBER K2 K) is sent to PUTON1 for inclusion on table I, and line (3) of the proof tree becomes (MEMBER (*PROD K1 K2 G) K). In mathematical language, to prove the desired implication, assume k_1 and k_2 are fixed, arbitrary members of K, and prove $k_1 k_2 \in K$.

Line (3) now has status PNT, a scan is of no help, and the line has no terms which are defined in the tables. The line's constants are K1, K2, G, and K, the first two of which are identified by table I statements headed by a term with a null definition. G is not identified at all, but K leads to a request for the definition of kernel. The user supplies the statement $a_1 \in K \iff f_1(a_1) \leftarrow e_H$ in ADEPT's language, and the appropriate instances of two implications appear on table I. The first of these is (IMPLIES (EQUAL

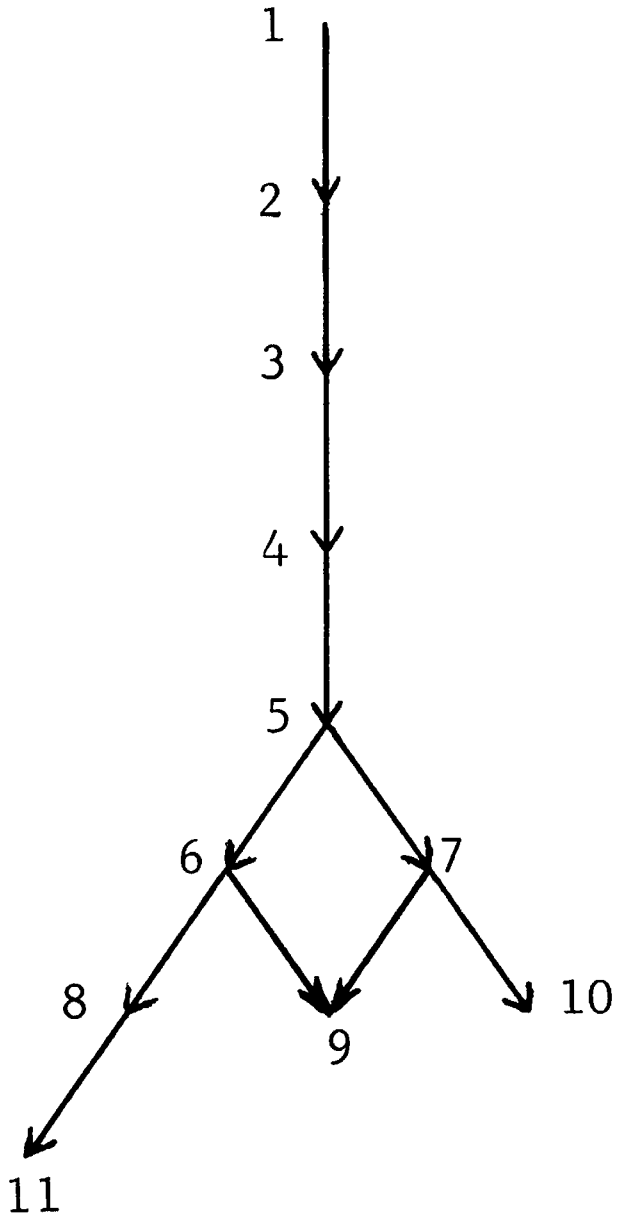


Figure 2

Proof tree for example

(F1 G H A1) (*IDENTITY H)) (MEMBER A1 K)), and the scan which takes place after the two entries are made to table I uses this to create the reduction (4): (EQUAL (F1 G H (*PROD K1 K2 G)) (*IDENTITY H)). In other words, to prove $k_1k_2 \in K$, prove that f_1 maps k_1k_2 into the identity. The scan and the progress determination are programmed so that the scan now stops and line (4) takes on status PNT. From it, an instance of the definition of identity is obtained and put on table I. (A two-sided identity is assumed by using the definition $a_1e \leftarrow a_1$ & $ea_1 \leftarrow a_1$). This leads to no reductions, but an examination of line (4)'s constants produces the "discovery" that (F1 G H) is a homomorphism. (Some special-case programming had to be inserted to provide that this be done, due to the prohibition on function symbols standing alone.) Upon obtaining the definition of homomorphism, the scan is able to produce (5): (EQUAL (*PROD (F1 G H K1) (F1 G H K2) H) (*IDENTITY H)). At this point the scan continues, using the other half of the definition of kernel along with other table I entries to derive facts about some of the objects in (5). Such facts are put on table I; in this case they are (EQUAL (F1 G H K1) (*IDENTITY H)) and (EQUAL (F1 G H K2) (*IDENTITY H)). Still continuing, the scan can now create from (5) the reductions (6): (EQUAL (*PROD (*IDENTITY H) (F1 G H K2) H) (*IDENTITY H)) and (7): (EQUAL (*PROD (F1 G H K1) (*IDENTITY H) H) (*IDENTITY H)), and from (6), (8): (EQUAL (F1 G H K2) (*IDENTITY H)) using the definition of identity, and (9): (EQUAL (*PROD (*IDENTITY H) (*IDENTITY H) H) (*IDENTITY H)). From (7) is obtained (10): (EQUAL (F1 G H K1) (*IDENTITY H)), and (9) is seen to be a reduction of (7) as well as of (6). From line (8) is obtained (11): (EQUAL (*IDENTITY H) (*IDENTITY H)) which is verified, as are all lines of the form (EQUAL A1 A1). This causes a chain of verification back to line (1), completing the proof. Thus ADEPT, using definitions of homomorphism, identity, and kernel, reduced $f_1(k_1k_2) = e_H$ to $f_1(k_1)f_1(k_2) = e_H$ and then to $e_H\ f_1(k_2) = e_H$, $f_1(k_1)e_H = e_H$, $f_1(k_2) = e_H$, $e_H = e_H$, $f_1(k_1) = e_H$, and finally to $e_H = e_H$, an obviously valid statement. The final proof tree has the form shown in Figure 2.

The use of EQUAL rather than EQUAL2 in the definitions of kernel and

identity is crucial in avoiding serious proliferation of lines. For instance, EQUAL2 in the definition of kernel would allow $f_1(k_1)e_H = e_H$ to have such reductions as $f_1(k_1)f_1(k_1) = e_H$, and EQUAL2 in the definition of inverse would lead to even worse misfortunes. To be sure, the above proof is itself inefficient, containing several irrelevant lines. However, the current version of ADEPT, whose development will be discussed in the next chapter, contains features which augment the basic structure shown in Diagram I and enable the program to avoid many needless actions. In this particular example, the proofs given by the old and latest versions deviate starting with the reductions of line (5), $f_1(k_1)f_1(k_2) = e_H$. In the improved procedure, substitutions of e_H for both $f_1(k_1)$ and $f_1(k_2)$ are made simultaneously and the subsequent reduction, $e_He_H = e_H$, is immediately simplified into the verifiable line $e_H = e_H$. Thus the current proof contains seven nodes, and the proof tree is a straight line. Certainly for that particular problem, that is a very efficient, naturally developing proof.

The scanning procedure is probably the most important subroutine in the program, and certainly it is the most complex. It is here that many heuristics can be added and tested. The main function of the scan, as already indicated, is to discover what new lines can be added to the proof tree as a result of implications on table I, and what substitution instances of lines already on the proof tree may be added to table II as a result of equalities on table I. This process is handled by a double loop, and can be done in either of two ways, both of which have been thoroughly explored with ADEPT. The first is to cycle through table I, and for each entry of one of the appropriate forms, to cycle through table II, generating all reductions of proof tree lines possible because of this particular table I statement. The other, of course, is to cycle through the proof tree, creating all reductions of a given line of table II due to all of the "known facts" on table I before proceeding to the next line of the proof.

The latter design of the scan is superior from the point of view of having the program achieve an "awareness" of where it is in a proof. The attention

of the program is not removed from, but remains focussed first and foremost upon the proof tree and its development. One consequence of this is the ease in which a concept of "related" substitutions can be implemented, permitting such steps to be made simultaneously and thus allowing such steps as the jump from $f_1(k_1)f_1(k_2) = e_H$ to $e_He_H = e_H$ which was seen to be desirable in the example just given.

The current version of the scan is known as SCANW, and it will now be explained in detail. It cycles through the proof tree in its outer loop, and contains such refinements as the related substitution feature. To understand its operation, the concept of logical class must be defined. (The reader will remember that an indicator of the logical class is the second element of a table I entry's property list.) The logical class represents an attempt by ADEPT to group together related table I entries. For instance, all conjuncts of a conjunction sent to PUTON1 are put in the same class (except for the conjuncts of the two conjunctions comprising the original hypotheses), and the two implications resulting from a biconditional are in the same class. Also in the same logical class are instances of the same definition added to table I during a single "explore consequences of all..." execution. Finally, any lemmas derived by the program, by procedures about to be described, from a table I implication will be in the same class as the implication. For example, if a line on the proof tree mentions two sets H and J, both of which are subgroups of a group G, and if both instances of the definition of subgroup are placed on table I during the examination of the constants of this line, then all conjuncts of both instances will be in the same logical class; in addition, if due to table I entries $a_1 \in H$ and $a_2 \in H$, the lemma $a_1a_2 \in H$ is derived using the conjunct $a_1 \in H \ \& \ a_2 \in H \Rightarrow a_1a_2 \in H$ of the definition of H, then this lemma will also be in the same class.

SCANW is described in Diagram II, which is augmented by Diagrams IIa - IIe, due to the routine's complexity. When studying these flow charts, note that this version of the scan contains within it a definition of "progress"; i.e., when the pointer is to be moved. This is by no means the only way to treat the

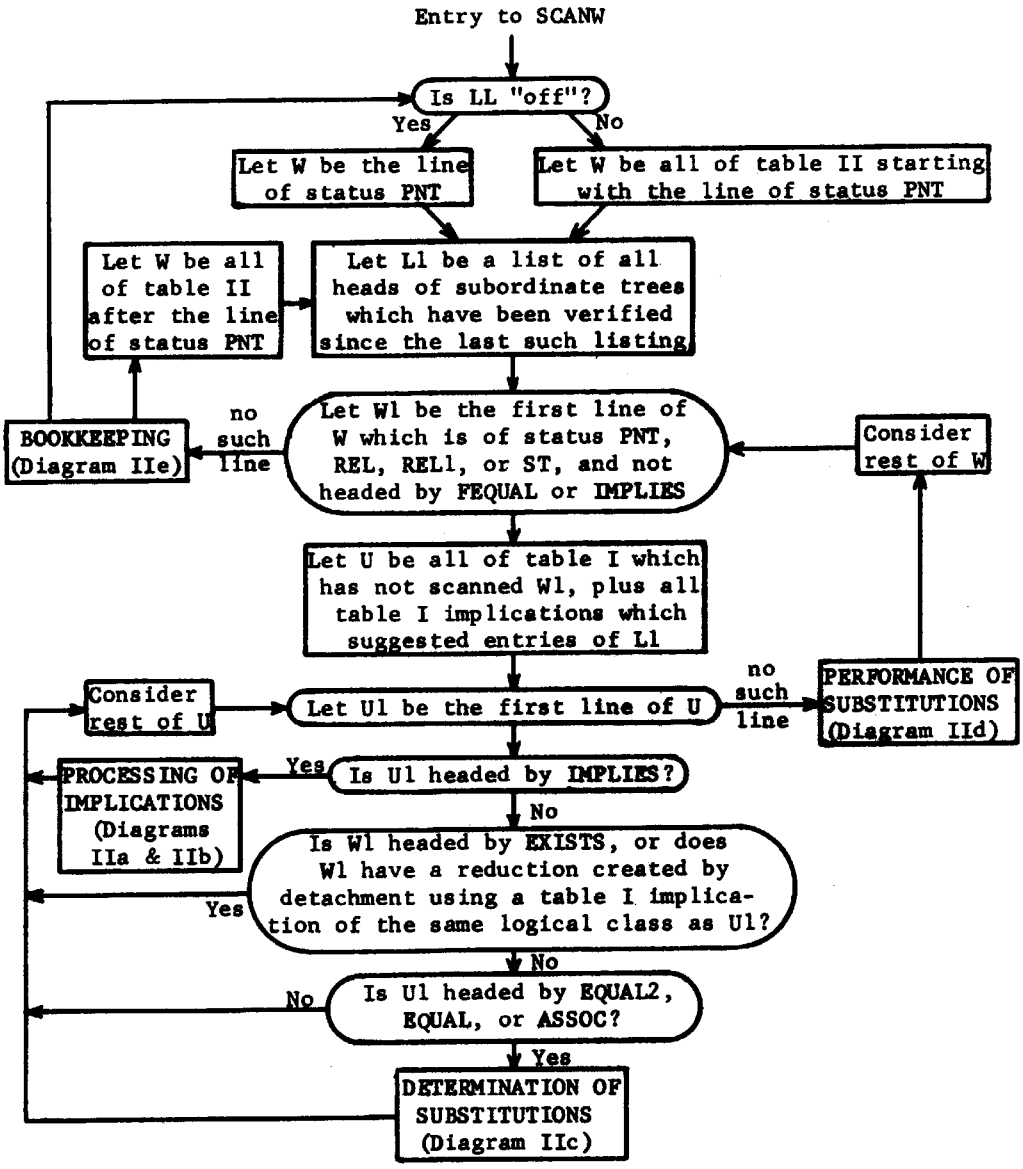


Diagram II

Flow chart for SCANW

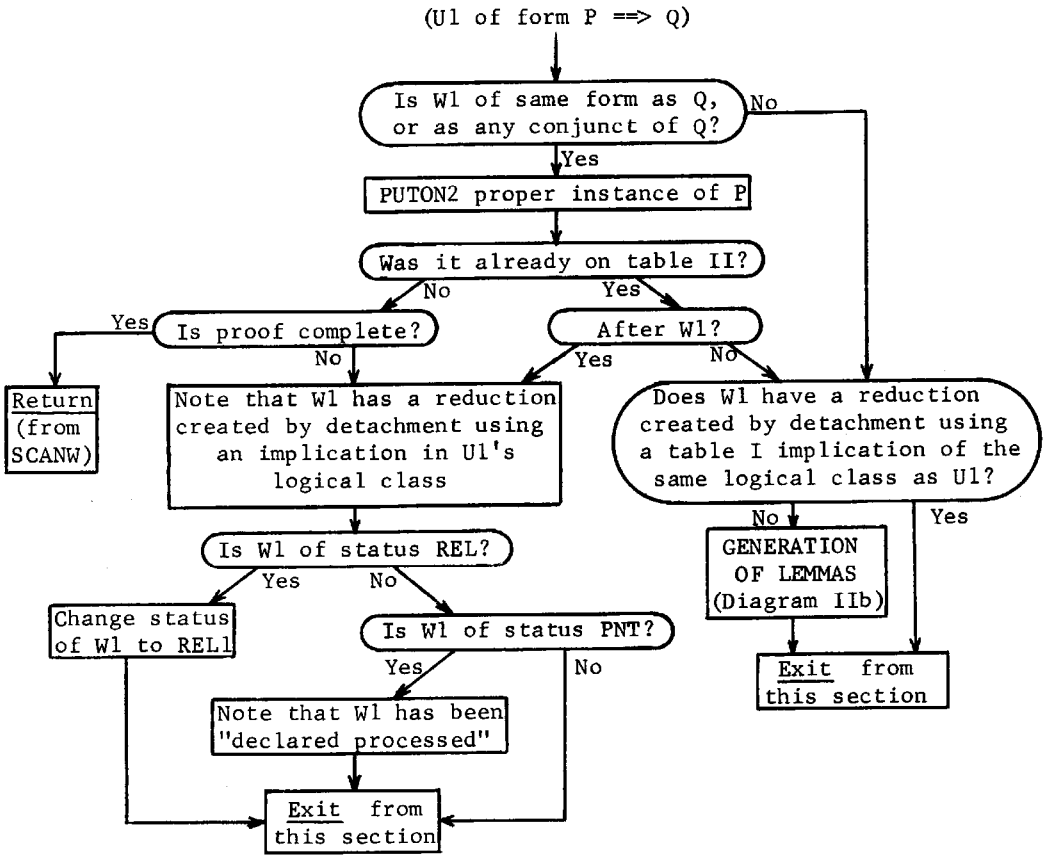


Diagram IIa

Flow chart for SCANW

(PROCESSING OF IMPLICATIONS)

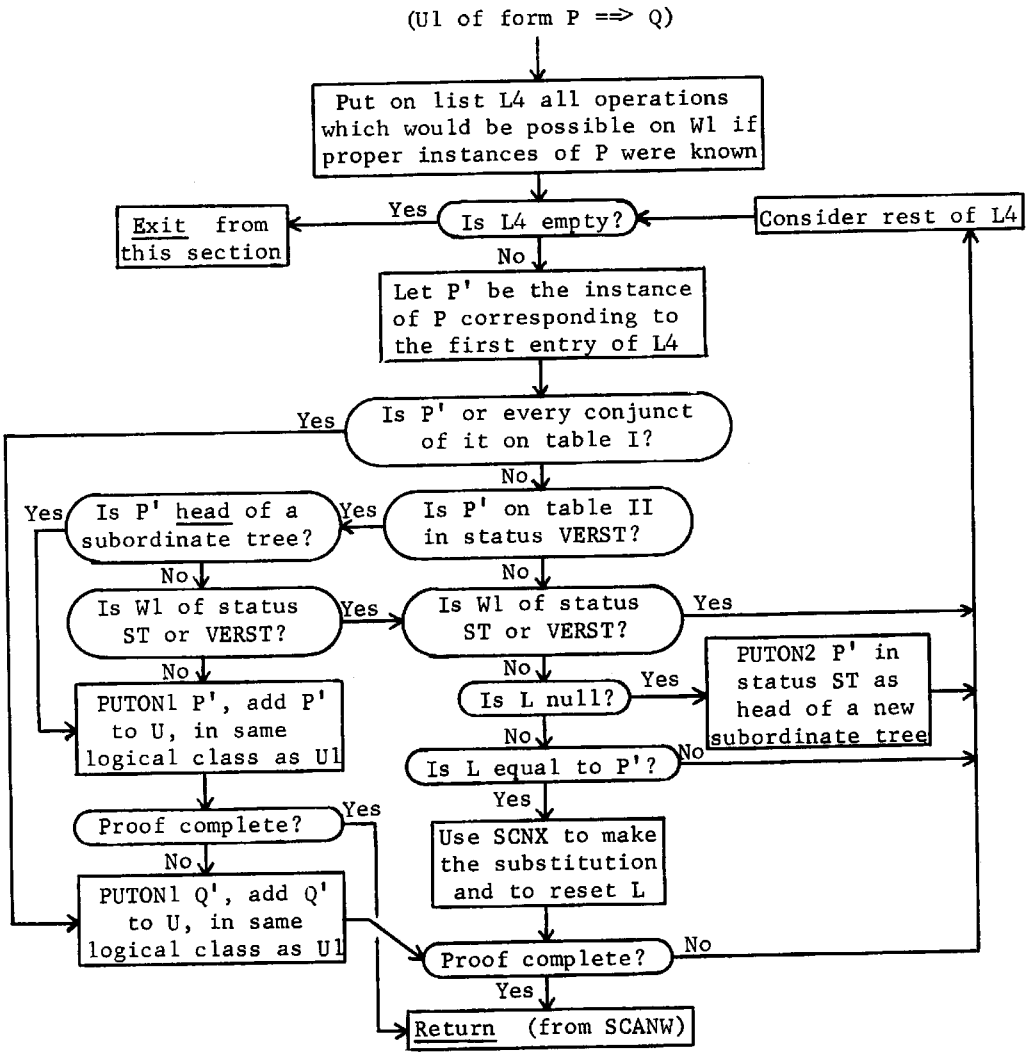


Diagram IIb

Flow chart for SCANW

(GENERATION OF LEMMAS)

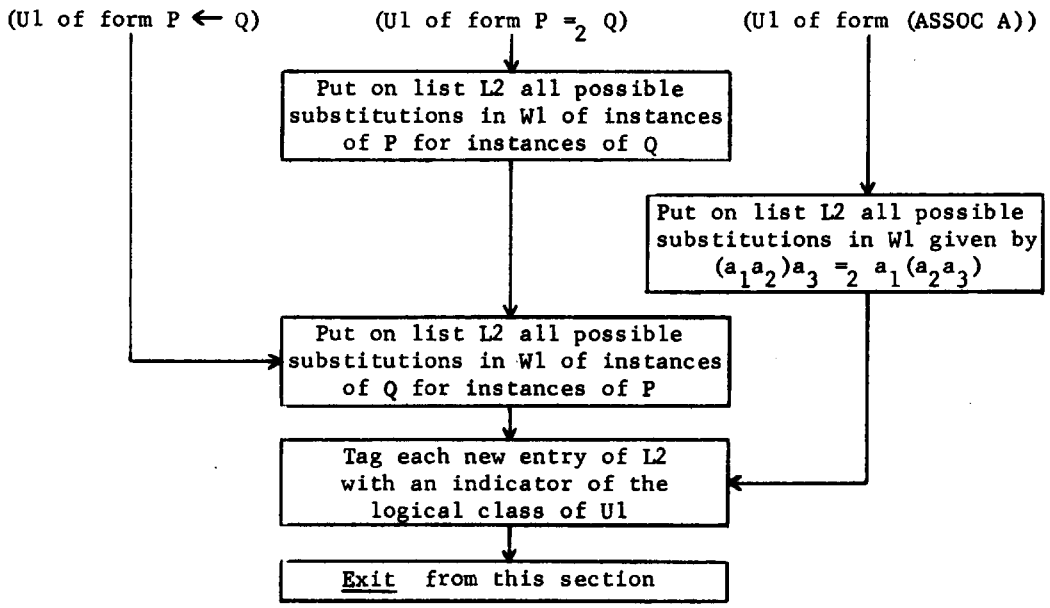


Diagram IIc

Flow chart for SCANW

(DETERMINATION OF SUBSTITUTIONS)

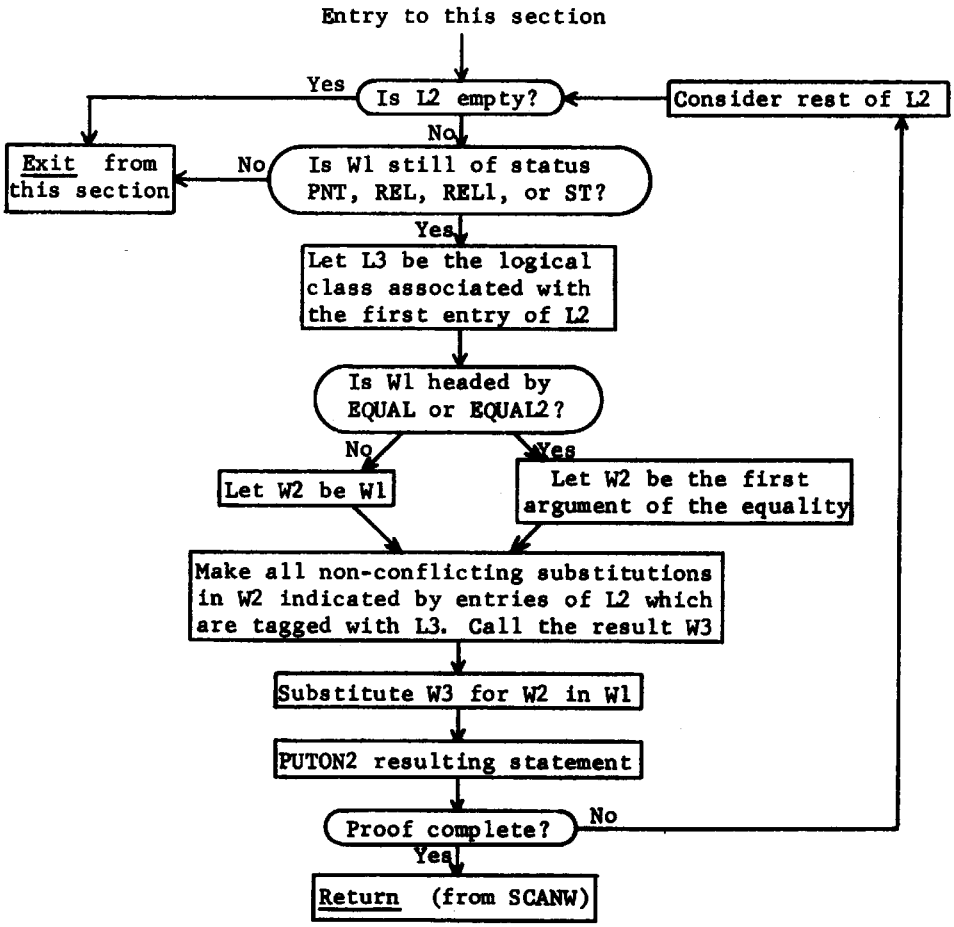


Diagram IId

Flow chart for SCANW

(PERFORMANCE OF SUBSTITUTIONS)

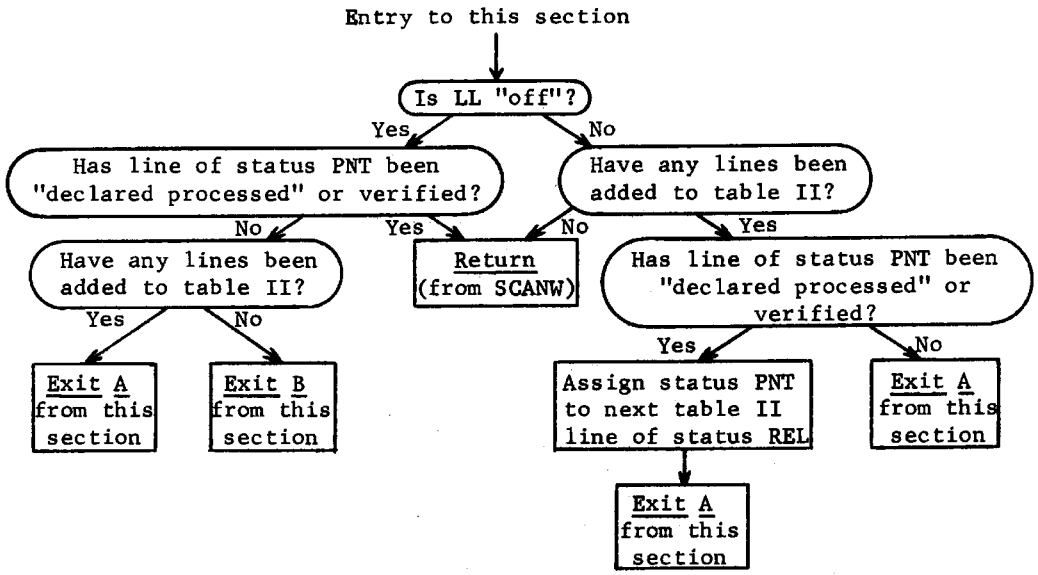


Diagram IIe

Flow chart for SCANW

(BOOKKEEPING)

evaluation of progress, and the next chapter will discuss this whole question in detail.

The scan, as a routine, has two arguments. One, L, is used only in the processing of implications. The second, LL, indicates whether all of the proof tree starting with the line of status PNT is to be scanned immediately, or if the node of status PNT is to be scanned, and then a decision made whether to continue or return (Diagram IIe). (The latter option is generally exercised when SCANW is used to "explore consequences". Other uses of the scan will be seen later.) As shown in Diagram II, before the scan is actually made, newly proved lemmas are noted, in order that whatever steps they make possible may be carried out. The generation and use of lemmas is part of the processing of implications (Diagrams IIa and IIb). The double loop is then entered, and implications and equalities are processed. Note that substitutions are not allowed in some proof tree lines which are scanned by implications, and that substitution instances are not actually created until the pass through table I is completed. This, of course, enables related substitutions to be performed together, and it is also a fact that some possible substitutions will not be performed due to actions taken while processing implications.

When an implication is encountered, as shown in Diagram IIa, an attempt is made to generate a reduction of the proof tree line being scanned by means of detachment. This method, used in Newell, Simon, and Shaw's LT, [14] is justified by modus ponens, and provides that a proof tree line P may be reduced to a line Q if there is an entry Q' ==> P' on table I, where P is an instance of P' (or of a conjunct of P') and Q is the corresponding instance of Q'. If an addition can be made to the proof tree by this means, certain bookkeeping is done, noting that the particular proof tree line has a reduction created in this manner. (The special treatment given to occurrences of the use of detachment is part of the current heuristic determination of "progress".) If not, and if the proof tree line being scanned does not have a reduction created by detachment from some other table I implication of the same logical class, then an attempt to generate lemmas is made. A lemma is

simply an instance of the consequent of the implication, which can be added to table I as a "known" statement by virtue of the presence of the corresponding instance of the antecedent (or all its conjuncts) on table I or as a verified line on table II. Thus the generation of a lemma is a direct use of modus ponens.

Quite often, a lemma which can be generated is irrelevant from the point of view of the statements to be proved on the proof tree. This suggested the inclusion of a relevancy check to be made before adding a lemma to table I. With this check, no such addition is made to table I unless the lemma would enable one or more reductions to be added to the proof tree. Once included, the existence of the check makes it natural to discover cases where a lemma would be useful but the required instance of the antecedent is not known to hold. Then this instance of the antecedent can be put on table II as the head of a subordinate tree, and if it is subsequently verified, the lemma will be generated by a future scanning pass if it still appears to be of use. In such an event the verified head of the subordinate tree is also put on table I. This is the present form of the section of SCANW which generates lemmas and which is detailed in Diagram IIb.

The restriction providing that lemmas are generated only when they can cause an addition to the proof tree is extremely important. It minimizes the use made of a "working forward" procedure and eliminates much unnecessary proliferation. For instance, suppose that G is known to be a group, and that a_1 and a_2 are members of G. From the definition of group it will be known that $e_G \in G$ and that G is closed under composition. Thus $a_1a_2 \in G$, $a_1e_G \in G$, $e_Ga_2 \in G$, and six other inclusions could be derived immediately from the closure statement, and each of these would lead to further "lemmas". With the present SCANW none of these derivations are made. (If one of these facts were desired; i.e., on table II and unverified, detachment would have handled the situation.) Suppose further that G is postulated to be abelian. Then none of the numerous instances of commutativity will be put on table I unless it enables a specific substitution to be made in a line of the proof tree. To

supplement the relevancy check, a heuristic, not indicated in Diagram II and which will be described in Chapter IV, has been found which limits the creation of some subordinate trees to those cases where the heads are "likely" to be verifiable.

Lines on subordinate trees are of status ST unless verified or simplified, in which case they are of status VERST or SIM respectively, though if a reduction of a line of status ST is already on table II, its original status will not be changed to ST. Lines of status ST are given a more restricted treatment than those on the main proof tree, since inasmuch as they cannot take on status PNT they can only be processed during scans. A second major restriction is that such a line cannot itself give rise to a new subordinate tree. This restriction to a single level of subordinate proof means that ADEPT cannot prove theorems requiring the establishment of a complex chain of subtheorems. Only a few of the problems considered required this kind of proof; to do them with ADEPT it was necessary to state some lemmas explicitly in the hypotheses.

One point is not clear on Diagram IIb, and this is the case where the argument L is not only not null (thereby prohibiting the creation of subordinate trees) but an actual statement. This situation arises when SCANW is called by the subroutine SOLVEX while attempting to obtain a "solution" for a table II line of the form $(\exists a_1)[a_1 \in A \ \& \ Pa_1]$. In such a situation, L is set to be "$a_1 \in A$". An example will be given when SOLVEX is discussed, later in this chapter.

The discussion of lemmas will conclude with an illustration. In this example a subordinate tree is started not because of an observed possible substitution (the most common case), but because of an observed possible utilization of an implication which is itself the consequent of an implication.

Theorem: If the <u>center</u> C of a <u>group</u> G contains all
of G, then G is <u>abelian</u>.

From the definition of abelian, ADEPT quickly concludes that the theorem will be proved if, assuming g_1 and g_2 are members of G, the identity

$g_1g_2 = g_2g_1$ can be established. This line itself suggests no continuation, but an examination of the terms of the first hypothesis causes an instance of the definition of center to be placed on table I, namely $[a_1 \in C \iff [a_2 \in G \implies a_1a_2 = a_2a_1]] \;\&\; [a_1 \in C \implies a_1 \in G]$. From this the scan finds that the implication $[a_2 \in G \implies g_1a_2 = a_2g_1]$ could reduce the proof to showing $g_2 \in G$, if it were known that $g_1 \in C$. This is not known, so $g_1 \in C$ becomes the head of a subordinate tree. The scan continues and finds that due to hypothesis a reduction of this line is $g_1 \in G$. This is an assumption made in the course of the proof, so the head of the subordinate tree is verified, and subsequently the desired implication appears on table I as a generated lemma. Continuing the scan, the proof is reduced to showing $a_2 \in G$, an assumption, thus establishing the theorem.

Diagrams IIc through IIe are mostly self-explanatory. Diagram IIc depicts the determination of substitutions, and shows clearly that the logical constant EQUAL suppresses substitutions in one direction. Note also that the associativity axiom, which can be stated using an equality headed by EQUAL2, can also be stated using a term of one argument, ASSOC (which is given a null definition), in order that the matching may be carried out by a more efficient, special-purpose subroutine. The actual creation of reductions which are substitution instances is shown in Diagram IId, and it should be observed that substitutions are only made in the left half, or first argument, of equalities on the proof tree. This particular heuristic, along with many others, will be discussed in Chapter IV. This concludes the exposition of the scanning subroutine.

The subroutines which add entries to table I and II, PUTON1 and PUTON2, are outlined in Diagrams III and IV. Both routines add an appropriate property list to the statement being added to a table, though the details of this bookkeeping are not all given in the flow charts. Note that both routines subdivide conjunctions and biconditionals, though PUTON2 also retains the original line and puts it on the proof tree in status SIM. Both diagrams contain references to the MODEL heuristic, to be introduced in the next

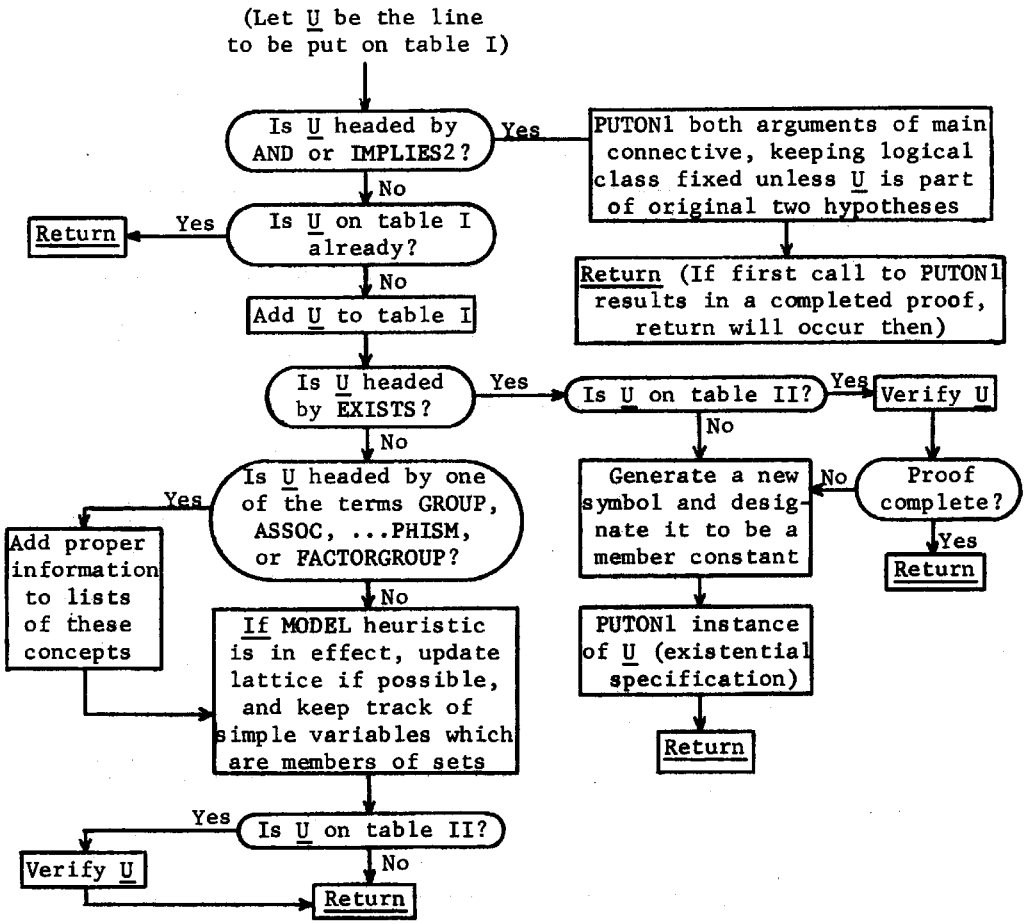


Diagram III

Flow chart for PUTON1

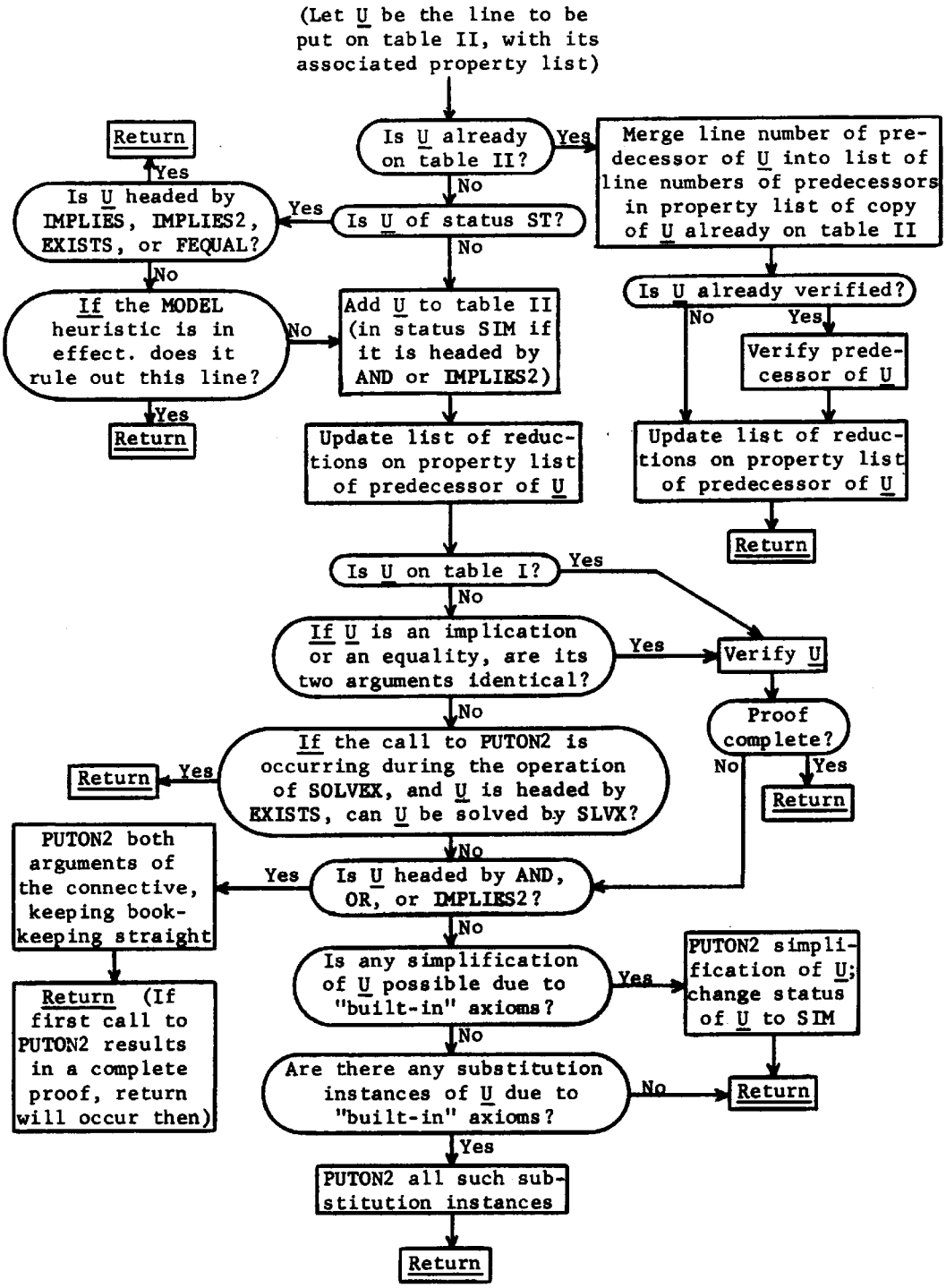


Diagram IV

Flow chart for PUTON2

chapter. This is the heuristic restricting creation of some subordinate trees.

PUTON1 compiles lists of certain information in order that searches of table I in its entirety may be held to a minimum. It is also clear from Diagram III that when a statement headed by an existential quantifier is added to table I, an instance of it is immediately derived and added to table I — a process corresponding to the classical operation of existential specification. (Existential generalization is not possible in ADEPT without the addition of a special axiom as part of the hypotheses.)

There is one step on Diagram IV which involves use of the subroutine SOLVEX and thus, like one step in SCANW, will be explained when SOLVEX is discussed. Then there is mention of "built-in" axioms, and these are a heuristic feature added as ADEPT was developed. As will be seen in Chapter IV, this feature has been very useful, but ADEPT could theoretically run with no such axioms.

Diagrams III and IV indicate the possible ways that a proof tree line may become verified, thus initiating appropriate tree-pruning by reference to the structure reflected in the lists of line numbers of predecessors and reductions in the property list of each table II entry. The simplest situation occurs when both arguments of an equality or implication being put on the proof tree are identical. Such a line is verified. Another case is when a line about to be put on table II as a reduction of some line n, is found to be already on table II and of status VER or VERST; i.e., verified. This causes appropriate tree-pruning starting with verification of line n. Otherwise, verification occurs only if a line appears on both table I and table II. This may come about in either of two ways. The line may be on table II and being put on table I as a newly-discovered "known" statement, or a line being put on the proof tree may already be on table I. The latter situation is extremely common, being the result of successfully reducing a statement to be proved to a statement already established.

For the purpose of checking whether or not a line is on both tables, the

different logical constants used to express equalities may be considered identical. While EQUAL and FEQUAL are used to achieve certain special processing of statements in which they appear, clearly their presence should not prevent verification of a table II equality which is on table I though expressed with a different logical constant. The heuristic use of the extra logical constants does not alter the fact that they all represent equality.

Diagram I contains a call to a subroutine called SOLVEX in order to process lines of the form $(\exists a_1)Pa_1$. This subroutine's effects have also been noticed in the flow charts for SCANW and PUTON2. It attempts to "solve" a line of the indicated form, for a value of a_1. As indicated in Chapter II, it is organized in terms of special cases. The current version will only handle Pa_1 of the form $P' =_2 P''$ or $P' \leftarrow P''$ where either P' or P'' but not both contains a_1, or Pa_1 of the form $a_1 \epsilon A \ \& \ Qa_1$ (or $Qa_1 \ \& \ a_1 \epsilon A$) where Qa_1 is one of the admissible Pa_1's just described. Such a line is put into the form $Q'a_1 = Q''$, and the conjunct $a_1 \epsilon A$, if any, is separated. The reordered line is checked to see if the equality can be solved for a_1 by "multiplying" both sides by proper inverses. For instance, $a_2 a_1 a_3^{-1} = a_4$ can be solved to obtain $a_1 = a_2^{-1} a_4 (a_3^{-1})^{-1}$. (Note that this subroutine assumes that all inverses are defined, and therefore cannot be used for theorems about semigroups. A check to determine the routine's applicability, though not included in the program at present, could easily be made using the list of sets known to be groups that is created by PUTON1.) If a solution is obtained the line either will be verified, or the indicated substitution instance of $a_1 \epsilon A$ added to the proof tree as a reduction of the line under consideration, whichever is appropriate. If no solution is found, the line is checked to see if it is of the form $Q'a_1 = Q'a_2$, in which case again it is either verified, or the reduction $a_2 \epsilon A$ added to the proof tree. If this also fails, a scan is done, with the arguments LL and L of SCANW "on" (i.e., not null), and if a conjunct $a_1 \epsilon A$ was detached from the line, L is set to precisely that conjunct. Whenever a new line of the form $(\exists a_1)P'a_1$ is put on the proof tree in the course of this scan, it is checked to see if it can be solved in one of the two ways mentioned.

This is the call to SLVX indicated in Diagram IV. If so, the scan ends just as if the proof were complete, and control returns to SOLVEX, which proceeds essentially as though the original line had been solved for a_1. All the lines created of the form $(\exists a_1)P'a_1$ during the scan are removed from further consideration; in fact, they are printed out at that time and completely removed from table II. If the scan ends without creating a line which can be solved, control returns to SOLVEX and then to the main routine which acts accordingly, as shown in Diagram I.

It remains to discuss the role of SCNX, mentioned in Diagram IIb. If during a scan initiated by SOLVEX, a substitution could be made in a line $(\exists a_1)P'a_1$ (i.e., in $P'a_1$) if only $a_1 \in A$, and L is "$a_1 \in A$", control is passed by SCANW to SCNX, which allows the substitution and resets L. This is best described by an example. Suppose that the statement $a_1 \in A \Longrightarrow (\exists a_2)$ $[a_2 \in B \ \& \ a_1 = a_2C]$ is on table I (one part of the definition of a factor group $A = B/C$), and that the line of table II being considered by SOLVEX is $(\exists a_1)$ $[a_1 \in A \ \& \ f_1(a_1) = a_3]$, where a_3 is being used as a constant. This will be "reordered" as $(\exists a_1)[f_1(a_1) = a_3]$ and L set to "$a_1 \in A$". (Clearly the scan will occur, as $f_1(a_1) = a_3$ cannot be solved directly.) The scan will observe that the substitution of a_2C (or more precisely, an instance of a_2C obtained after an application of existential specification) for a_1 could take place if $a_1 \in A$ were established. SCNX receives control and adds the line $(\exists a_2)$ $[f_1(a_2C) = a_3]$ to the proof tree, sets L to "$a_2 \in B$", and saves the information that this substitution has been made. This suffices to illustrate the operation of SCNX, but it will be profitable to continue this example further. Perhaps f_1 is a map from B/C to D/E defined by $f_1(a_1C) = f_2(a_1)E$, for some map f_2 taking B <u>onto</u> D. So, if $a_3 \in D/E$, the lines $(\exists a_2)[f_1(a_2C) = a_4E]$, $(\exists a_2)[f_2(a_2)E = a_4E]$, and $(\exists a_2)[f_2(a_2)E = f_2(a_5)E]$ might be generated. The last of these can be solved, giving $a_2 = a_5$ (for this possibly non-unique a_5 such that $f_2(a_5) = a_4$, where $a_3 = a_4E$), and SCANW will now return control to SOLVEX, which will make use of the information stored by SCNX that a_2C was substituted for a_1, and put on the proof tree as a reduction of the original

line $(\exists a_1)[a_1 \in A \ \& \ f_1(a_1) = a_3]$ the line $a_5 C \in A$. This "change of variables" procedure, which may be carried to any depth, is clearly very useful.

In Diagram I there is also a reference to a subroutine HOMOMF, which deals with lines headed by FEQUAL. (The name HOMOMF was chosen because the routine is used in establishing homomorphisms.) HOMOMF's task is to attempt to verify the equality headed by FEQUAL by successive evaluation of the two arguments of the logical constant; i.e., the two sides of the equality. It evaluates the first argument, obtaining, if successful, a list of one or more evaluations, plus the original argument. It then starts to evaluate the second argument, stopping with success if an evaluation of the second argument is on the list of evaluations of the first. Evaluation is done by a call to SCANW, with table II augmented by a new "tree" headed by the argument to be evaluated. The use of FEQUAL and HOMOMF has proved to increase efficiency in the establishment of equalities similar to the one involved in demonstrating that a map is homomorphic. As an illustration, suppose that table I contains the hypothesis $f_1(a_1) = e$; i.e., f_1 maps everything into the identity. If a sufficient condition for a map's being a homomorphism, stated using FEQUAL, is entered into ADEPT's tables, then in the course of proving f_1 homomorphic HOMOMF will be asked to establish the equality $f_1(a_2a_3) =_f f_1(a_2)f_1(a_3)$. The proof would proceed: $f_1(a_2a_3) = e$ (evaluation of the first argument); $f_1(a_2)f_1(a_3) = ee = e$ (q.e.d.).

It is possible to write a supervisor to use the full power of ADEPT to solve a theorem with many sub-theorems. Such a program is ISOLVE, which uses ADEPT to establish isomorphisms between groups. A routine called GENFCN was written, which generates a canonical relation between two sets, given the hypotheses concerning the two sets (GENFCN also uses SCANW), and ISOLVE makes four calls to ADEPT to prove that the relation is (in this order) well-defined, homomorphic, epimorphic, and monomorphic. (Sometimes the relation generated by GENFCN is well-defined by construction, as in the case when the domain of the relation is simply a group G.) This program has been used to establish a number of isomorphisms, and these are the most impressive theorems proved by

the computer in the course of this project. (Incidentally, GENFCN works for sets but ISOLVE assumes that the sets are groups.)

One fact about ISOLVE is of special interest. In order to prove a relation f_1: G $\longrightarrow$ H is well-defined, the problem $a_1^{-1}a_2 = e_G \Longrightarrow f_1(a_1)^{-1}f_1(a_2) = e_H$ is given to ADEPT. In general, to prove that f_1 is one-to-one requires establishment of the converse implication. However, since f_1 will first be shown to be a homomorphism (if f_1 isn't proved to be homomorphic, or if any one of the four proofs is not successfully completed, ISOLVE goes no further), one may take advantage of the theorem which states that for a homomorphism f_1, f_1 is one-to-one if and only if its kernel reduces to the identity. That is, ISOLVE is justified in presenting ADEPT with the problem $f_1(a_1) = e_H \Longrightarrow a_1 = e_G$, and in fact, this is exactly how it has been programmed to prove f_1 monomorphic, for this latter implication usually can be proved more easily. Equivalently, ADEPT could be programmed to include the heuristic — "when asked to prove that a function f_1 is one-to-one, make use of (the above theorem) whenever it is known that f_1 is a homomorphism".

CHAPTER IV

SPECIAL-PURPOSE HEURISTICS

THE DYNAMIC PROCESS OF USING AND CREATING ADEPT

The heuristic philosophy adopted for the ADEPT project has already been expounded, and certain of the program's heuristic features have become apparent in the previous chapters. The original version of the program contained relatively few heuristics, inasmuch as the purpose of the development of ADEPT was to discover the difficulties which arise in the attempt to prove theorems of group theory by computer, and then to devise means by which to overcome these obstacles. Thus it is pertinent to discuss not only the current version of the program, but also the development of this version, and some of the previous states of the program. By this, of course, is not meant a revelation of such data as the state of the program when the ability (SOLVEX) to handle problems involving certain statements headed by an existential quantifier was added, but rather a discussion of heuristics and features which were developed to cope with previous lack of capability or efficiency on the part of ADEPT. Thus this chapter supplements the previous one in giving further description of the program, concentrating on those features whose addition was prompted by information obtained while experimenting with early versions of ADEPT.

One such addition has been described in Chapter III, since it is presently an important part of the operation of SCANW. This is the simultaneous performance of related substitutions, a heuristic which eliminates many previous instances of inefficiency in favor of very natural steps combining operations.

Another good example, and a simple one, is reflected in Diagram I, and happens to represent the first difficulty which arose in this project. Initially, the program only put instances of definitions on table I as a result

of processing the line of status PNT. But, as is shown in Diagram I, when ADEPT has explored the consequences of all such instances without success, it now considers all terms in the first hypothesis of the problem and obtains the appropriate definition instances for them. Such action was not necessary in the proof that the kernel of a homomorphism is a submonoid, for instance, but it is required in order to prove that the center of a group is abelian. In this example, line (1) of the proof tree, (ABELIAN C G) is reduced to the implication which is the indicated instance of the definition of abelian, which is promptly split, thus obtaining line (3), (EQUAL2 (*PROD C1 C2 G) (*PROD C2 C1 G)) which becomes the first line of status PNT to be searched for terms and constants. But no explicit mention of C appears in this line, and since the proof requires the addition of an instance of the definition of center to table I, the examination of the terms of the first hypothesis takes place, and the proof is subsequently completed.

Note that the implementation of this heuristic assumes the use of non-asterisked terms to some extent. For instance, if (CENTER C G) had not been included in the hypotheses of the above example, but instead the conclusion had been stated (ABELIAN (*CENTER G) G), the heuristic as programmed would have failed. To remedy this, and to implement the heuristic in a more general form which will have the same effect independent of notation, all asterisked terms of the conclusion should be examined along with the terms of the first hypothesis.

The difficulties which suggested this heuristic reflect a second need for augmenting the "working backward" approach with a certain amount of "working forward". (The generation of lemmas, of course, is the first and primary example.) In other words, a procedure which only reduces desired conclusions to established statements is not adequate for handling theorems of group theory. Some recourse must be made to the derivation of statements from the hypotheses, axioms, etc. Table II, the proof tree, cannot develop sufficiently without such derivations, which cause expansion of table I. However, such expansion must be controlled. Lemmas are generated only if they will be

of immediate use. In the same spirit, it is important to note that a check of terms in the hypotheses is not done until the usual procedures prove inadequate for a given problem.

The whole question of when to obtain what information from the condition and definition tables is a problem of information retrieval. In harder theorems requiring a larger base of information, this problem could become quite critical. Some comments on this, and some alternative methods of placing and using definition instances on table I, will be given in Chapter VIII.

Another instance of a change which was made in the course of the development of ADEPT was also seen in the flow diagrams of SCANW, in particular, in Diagram IId. This is the heuristic which provides that reductions obtained by substitutions in an equality on table II are created by substituting only in the first argument of the equality. In other words, if a line on the proof tree has the form $P = Q$, substitutions are made only in P. The rationale behind this restriction is illustrated by an example. If it is known explicitly that $P = P_1$, $P_1 = P_2$, ..., $P_{n-1} = P_n$, $P_n = Q_n$, ..., $Q_1 = Q$ (where none of the expressions are identical, and no other equivalences are as yet explicitly "known"; i.e., on table I), a proof of $P = Q$ with the heuristic under discussion takes 2n steps. If substitutions are permitted in Q also, and no other heuristic is introduced to order substitutions, a proof of $P = Q$ might proceed as follows:

$$\begin{array}{ccccccc} P = Q & \longrightarrow & P_1 = Q & \longrightarrow & P_2 = Q & \longrightarrow & \\ \searrow & & \searrow & & \searrow & & \\ P = Q_1 & \longrightarrow & P_1 = Q_1 & \longrightarrow & & & \end{array}$$

a process which will require on the order of n^2 steps.

In return for the potential reduction of the number of steps, or at least for release from the need to discover more sophisticated heuristics, there would be, in theory, no cost if all equalities were treated symmetrically. But such is not the case in ADEPT when use is made of EQUAL. However, even if all equalities were headed by EQUAL2, there is a price to be paid in practice. Consider the proof that the map which takes every element of a group G into the identity is a homomorphism. This proof was discussed in the previous

chapter as an example of the use of FEQUAL. However, suppose the equality expressing the property of being homomorphic is stated using EQUAL2. Then the heuristic of substituting only in the left-hand side requires the proof to proceed: $f_1(g_1g_2) = e = ee = f_1(g_1)f_1(g_2)$. But this series of steps, using only the hypothesis $f_1(a_1) =_2 e$ and the definition of identity, requires planning heuristics of a comparatively involved nature. For instance, to reduce the line $e = f_1(g_1)f_1(g_2)$ to the line $ee = f_1(g_1)f_1(g_2)$ requires expansion of e to ee, which might be accomplished upon analysis of the right-hand side of the equality. Even more difficult to program would be the efficient execution of the substitutions needed to establish the final result. (None of the expansions would normally be allowed, since the hypothesis defining f_1 would most naturally be stated using EQUAL, not EQUAL2, as would the definition of identity.) Admittedly, this problem is primarily an example of the worth of the use of FEQUAL, but it does show the kind of difficulty which can arise due to the restriction on substitutions in equalities on the proof tree.

There is one heuristic which was included in the original program, and which has been retained because of its success. This particular feature is a restriction included in ADEPT's matching subroutine, and thus was not encountered in Chapter III. Incidentally, the matching process used is basically recursive and no attempt has been made to optimize it, yet it suffices quite well. The restriction concerns the case where a single variable is being matched against a line or part of a line on the proof tree. In other words, all occurrences of objects, of the same type as the variable, appearing in some table II line are desired. In this event, the variable is matched only with simple variables or constants, not with compound objects. This case is not to be confused with the matching of variables within a more complex expression being matched against a line of the proof tree; such variables will be matched with any object of the appropriate type. Thus, if all reductions made possible by a table I entry (EQUAL A1 (*INVERSE A1 G)) are desired from the line (MEMBER (*PROD A2 A3 G) H), substitutions of a_2^{-1} for a_2 and a_3^{-1} for

a_3 will be made, but the substitution of $(a_2a_3)^{-1}$ for a_2a_3 will not be considered. However, the A1 in an expression such as (*PROD A1 A2 G) may be matched with any member object.

This heuristic has the obvious effect of controlling the expansion of a proof tree in any problem in which it is applicable. In only one theorem presented to ADEPT did it cause any unwanted effects, so it is certainly to be considered advantageous. (The one difficulty necessitated the explicit definition of the inverse image of a point, as opposed to the use of the general definition of the inverse image of a set in conjunction with the definition of a unitset or singleton; i.e., a set with one member.)

Perhaps the most important and revealing addition to the program is one which occurred in stages. As alluded to in the previous chapter, the first implementation of ADEPT had no identities or definitions "built into" the program. Thus simplification of a product involving the identity element could not be accomplished until the term *IDENTITY was encountered and the corresponding instance of its definition put on table I. Also, certain facts, such as "$f_1(e) = e$ when f_1 is a homomorphism", had to be included with the hypotheses for some problems, for ADEPT is not able to set up the "constructions" necessary to prove these facts. Clearly, this forced the program to be quite inefficient in proving theorems. Such a simple-minded procedure does, however, emulate the student who is a complete beginner at group theory.

Just as a student must assimilate simple identities involving identities, inverses, homomorphisms, etc., it does not take long before the problems in group theory demand facility with these facts if efficient proofs are to be found. It was clear that ADEPT could not become proficient and handle harder problems unless these identities could be applied more sensibly. Accordingly a routine was introduced (into PUTON2, see Diagram IV) to check each line being put on the proof tree to see if certain identities could be applied. As the number of theorems presented to ADEPT increased, it became clear which identities should be dealt with in this way, and the routine under discussion expanded gradually. Besides the advantage of a more natural development of

the proof tree, this routine offered an opportunity to make this use of matching procedures more efficient. This is so because the identities to be matched are not arbitrary, but fixed, and a special-purpose matching routine can be easily implemented and used instead of the general matching routine used by SCANW. In ADEPT's most recent version, matching is greatly speeded up by a key-word check between the identity and the line of the proof tree.

The current set of identities handled in this way is divided into two classes, according to the action taken if a match is found. The first set causes a true simplification; i.e., a new line is created as a reduction of the old line, and the status of the original line is changed to SIM. This set contains the facts $ee \leftarrow e$, $ea_1 \leftarrow a_1$, $a_1e \leftarrow a_1$, $a_1a_1^{-1} \leftarrow e$, $a_1^{-1}a_1 \leftarrow e$, $(a_1^{-1})^{-1} \leftarrow a_1$, and $e^{-1} \leftarrow e$. There is also a subroutine by which products of arbitrary association are checked according to these identities. Thus the product $(a_2a_3)a_3^{-1}$ will be simplified to a_2 even though the binary nature of the operation does not allow direct application of the identity $a_1a_1^{-1} \leftarrow e$. (This subroutine obviously relies upon the assumption of associativity.)

The second set of identities merely causes new lines to be put on the proof tree, without declaring the original line unworthy of further consideration. In this set presently are the following facts: $(a_1a_2)^{-1} =_2 a_2^{-1}a_1^{-1}$; if f_1 is homomorphic, $f_1(e) \leftarrow e$, $f_1(a_1^{-1}) =_2 f_1(a_1)^{-1}$, $f_1(a_1a_2) =_2 f_1(a_1)f_1(a_2)$; if A/B is a factor group, $a_1^{-1}B =_2 (a_1B)^{-1}$, $(a_1a_2)B =_2 (a_1B)(a_2B)$. Application of these identities is expediated by the lists of factor groups and homomorphic maps created by PUTON1.

It is revealing to note the analogy here between what is essentially a transition from interpretive to compiled application of these identities with the corresponding development in the methods used by a student in making such steps.[12]

The problem of deciding when a line on the proof tree has been successfully processed, even though the theorem may be not yet proved, is a very difficult one. For a general theorem-prover, this problem is closely related to the question of deciding which branch of the tree it would be most

profitable to investigate first. In ADEPT, the problem is sharpened; from a given line on the proof tree reductions should be generated until and only until the one which is "correct" is generated. This is one manifestation of the goal of having the program appear as though it understands where it is in the course of a proof. It turns out that there are heuristics which are fairly successful in this regard for a wide range of theorems. Unfortunately, their effectiveness decreases rapidly when irrelevant hypotheses are introduced. Without any heuristics, even problems with a minimal set of hypotheses have inefficient proofs, but while the schemes about to be discussed usually remedy that situation, they are of limited worth with regard to overspecified problems.

The "progress" heuristics used by ADEPT are few, and range from trivial to subtle and controversial. In the first category are rules such as considering a line on the proof tree of the form P ⟹ Q processed when an instance of the antecedent P is put on table I and the corresponding instance of Q becomes the reduction of the implication. In fact no other processing is allowed on such a line except for verification if the same line P ⟹ Q appears on table I. Similarly, a line headed by EXISTS or FEQUAL is processed only by the special-purpose subroutines SOLVEX and HOMOMF respectively.

The major difficulty occurs with lines of a general nature, e.g., equalities or lines of the form $a_1 \in A$. If such a line is not verified, it never can be declared to be sufficiently processed unless some heuristic is employed. Two have been tried with ADEPT. Both provide for "progress" only after a line of the proof tree has had a reduction introduced by detachment. But not all such reductions cause "progress" to be declared. A detailed description of the heuristics will make this clearer. Suppose that line n of the proof tree is under current consideration; i.e., of status PNT. Heuristic A provides that when the scan is called, all entries of table II from line n onward shall be processed, and as soon as the line of status PNT gives rise to a reduction by detachment it shall be declared to need no further processing if and only if the line of status PNT is the lone reduction of its immediate predecessor

(not counting co-conjuncts if the predecessor is a conjunction). If this is the case, the next table II line of status REL immediately takes on status PNT and the scan continues. Any line not of status PNT which obtains a reduction using detachment is not given any special treatment. Heuristic B provides that first only line n shall be scanned, and after it is completely processed by all table I entries, the scan shall stop if line n has given rise to one or more reductions via detachment. What is more, after an instance of detachment occurs due to a table I implication, no further table I equalities in the same logical class as the implication will be used to create substitution instances of line n. If detachment occurs, once the scan of line n is completed control will be returned to the main routine, which will move the pointer down; i.e., "progress" will be declared. If no detachment occurs, the scan will continue through all the rest of table II, and any line of status REL giving rise to a reduction using detachment will be put in status REL1, so that it cannot take on status PNT at any later time. The restriction on creation of reductions by substitution after instances of detachment also remains in force as the rest of table II is processed.

Heuristic A is clearly syntactic, being based on the structure of the proof tree preceding line n. It proceeds on the assumption that if the proof tree has been developing in a straight-line fashion, it is safe to proceed as though it were inevitable that the proof would continue in this fashion. Conversely, if the proof is full of alternative branches, the heuristic says that caution must be observed. It is clear that irrelevant hypotheses render heuristic A ineffective, for extra table I entries cause excess branching in the proof tree, unnecessary though it may be, and thus heuristic A will not cause "progress" to be declared as often as it should.

Heuristic B is the one currently in use in ADEPT, as was seen in Diagram II. It also gives special consideration to occurrences of detachment, but it does not refer to the shape of the tree. Unlike heuristic A, it takes notice of applications of detachment to lines other than the one of status PNT. Since it prevents some lines from ever taking on status PNT, unnecessary uses of

detachment can be harmful. Thus heuristic B is also adversely influenced by unnecessary hypotheses, but only if they contain implications. This slightly greater degree of insensitivity is of some advantage, and another factor in heuristic B's favor is the greater amount of processing allowed on a line of status PNT, so that it can continue to give rise to reductions even though one reduction of it has been created using detachment. To allow total processing to continue would be too inefficient, however, so the restriction involving logical classes has been introduced. An example will clarify the reasoning behind heuristic B, and the theorems which will be discussed in the following chapters will serve to indicate shortcomings of this strategy. In particular, the third part of the first example of Chapter VI shows the one situation encountered to date where the logical class restriction fails completely.

Consider the theorem stating that the image I of a group G under a homomorphism f_1 is a subgroup of the range of f_1. In the course of proving this theorem, ADEPT must show closure of the image under composition; in particular, having fixed b_1 and b_2 as members of I, it must be shown that b_1b_2 is a member of I. The relevant instance of the definition of image is $a_1 \epsilon\ I \Longleftrightarrow (\exists a_2)[a_2 \epsilon\ G\ \&\ f_1(a_2) = a_1]$, which becomes two implications on table I in the same logical class. One of these causes an occurrence of detachment, reducing $b_1b_2 \epsilon\ I$ to $(\exists a_2)[a_2 \epsilon\ G\ \&\ f_1(a_2) = b_1b_2]$. This happens to be the "correct" step, leading to a proof of minimal length, and is but one instance of advantageous applications of detachment which have led to the affirmation of a connection between detachment and "progress", as shown in the two progress heuristics. Now, the converse implication on table I leads to a substitution instance of $b_1b_2 \epsilon\ I$ that could be put on the proof tree, namely $f_1(g_1)f_1(g_2) \epsilon\ I$, where g_1 and g_2 are members of G which are mapped by f_1 into b_1 and b_2 respectively. This branch would lead to a parallel proof, doubling ADEPT's efforts, and it is this kind of proliferation which the logical class restriction was introduced to prevent. Indeed, the vast majority of the time this particular feature of heuristic B achieves exactly the proper effect. (Note that the success of this procedure depends upon the

detachment's being made before the substitutions are determined. This will be insured if the user puts all definitions which specify sets by their members in the form $a_1 \in A \iff Pa_1$, for PUTON1 splits a biconditional $P \iff Q$ into $Q \Rightarrow P$ and $P \Rightarrow Q$, in that order.)

One more heuristic remains to be discussed in this chapter. It is a non-trivial one, though easy to implement, and it is present because of the nature of many definitions, namely, that they are of the form $a_1 \in A \iff Pa_1$, where Pa_1 contains an equality of the form $a_1 = Q$, for some object Q. When an instance of such a definition is on table I, it is observed many times during a call to SCANW that a substitution of Q (or more precisely, some instance of Q) could be made for some variable or constant, say a_2, if only $a_2 \in A$ were established. Consequently a great many subordinate trees are started on table II, headed by a node of this form. Many of these nodes represent highly unlikely possibilities, and some are not just impossible to prove, but are actually incorrect. To remedy this, a lattice or model of the sets involved can now be built by ADEPT using information stored with the definitions of various kinds of sets. This lattice is used only to cull out unlikely possibilities about to be put on table II in status ST. It is not used for the purpose of verifying a line on the proof tree, in order that all proofs may be developed by the operations on the statements on tables I and II. An example will show what this heuristic does and does not do. With the definition of a factor group A = B/C is stored the following information: A, $B \supset C$. This indicates that C is a subset of B, and that B and A have no members in common. (The definition of a map f_1: A $\longrightarrow$ B is accompanied by the information A, B. This indicates that there is no reason to assume that the sets A and B are not disjoint.) Similarly for a sub-factor group D = E/G of a factor group I/G, the definition is augmented by the information D, $I \supset E \supset G$. So if the hypotheses of a problem provide that A = G/H, B = G/K, C = H/K, D = B/C, and $I \subset G$, ADEPT can construct the model D, A, $B \supset C$, $G \supset H \supset K$. This lattice

$\cup$
I

will be used in conjunction with other hypotheses about variables and

constants. For instance, suppose a_1 is known to be in H. Then the statement $a_1 \in B$ will be ruled out by a simple reference to the lattice. $a_1 \in K$ will be discarded as unlikely, for a general member of H can seldom be shown to be a member of a particular subset of H. $a_1 \in G$ will be allowed, but not automatically verified. $a_1 \in I$ will be allowed, since it is quite possible, and such a statement is often provable in group-theoretic problems (as opposed to $a_1 \in K$).

This heuristic only examines lines of the form $a_1 \in A$ where a_1 is a simple variable or constant, as opposed to a compound object such as a product. This is so mainly because of the difficulty of formulating and applying rules for accepting or rejecting the more complex statements. For instance, suppose that a_1 and a_2 are both members of G and H is a subset of G. Even though it is reasonable to exclude statements such as $a_1 \in H$ with almost any specification of H, as long as it is a proper subgroup of G, it is not always wise to exclude the statement $a_1 a_2 \in H$. To be sure, often this is also a fruitless head for a subordinate tree, but H might be a set such as the coset $a_1 J$, where J is some other subset of G. Another restriction on the heuristic's applicability is that it is not used to examine proposed lines for the main proof tree. If it were, it would be impossible to use ADEPT to prove such facts as "the center C of an abelian group G is the whole group." In this problem, it is known from the definition of center that C is a subset of G, and it must be shown that any member a_1 of G is a member of C. Clearly the heuristic must not be allowed to prevent the necessary conclusion from being placed on the proof tree! Fortunately, the heuristic serves its purpose quite adequately, though it is restricted to apply only to lines of the form $a_1 \in A$ about to be put on subordinate trees, where a_1 is a simple variable or constant.

This feature, known as the MODEL heuristic, was implemented so that it can be used or not as desired. Tests with isomorphism problems have shown a steady decrease of about 1/3 in the length of a proof, with all lines discarded being indeed irrelevant. The decrease in processing time was nearly the same, being only slightly less, due to the nature of the information to be handled

in the lattice, which allows very fast processing of this heuristic. Clearly because of the nature of many definitions in group theory and modern algebra in general, this is an important special-purpose feature.

Though the MODEL heuristic is an important addition to ADEPT because the single operation it performs is so often desirable, it is not a profound example of the use of a semantic model. Such models are not uncommon in theorem-proving. A simple example would be the augmentation of a propositional calculus theorem-prover with a subroutine which tested a proposed line by evaluating it using one of the possible assignments of truth-values to its variables. The best-known example of the use of a semantic model is the "diagram" in Gelernter's geometry-theorem proving machine.[5,6] This program and its use of a model is discussed in Appendix I. In general, heuristic programs for proving theorems often can be improved by inclusion of some model of the intended interpretation of its domain, to act as a filter or guide for using the more syntactic procedures in the algorithm.

Other insights have been gained during the course of this project. Because they did not lead to the addition of important heuristics, and because they are best discussed in the light of particular examples, they are not considered here, but postponed until the next chapter, which discusses selected theorems successfully proved by ADEPT.

CHAPTER V

DISCUSSION OF SELECTED PROBLEMS

ADEPT has been used successfully to prove nearly 100 theorems. To be sure, some of these have been very simple problems, but many have been of a relatively involved nature. 17 have been proofs of isomorphisms by ISOLVE, each of which involves at least 3 if not 4 separate sub-theorems (depending on whether or not the relation requires explicit proof that it is well-defined). The greatest number of nodes in a proof tree was 73 (see the third problem discussed in this chapter), and no single proof by ADEPT took more than $6\frac{1}{2}$ minutes of computer time, though the composite isomorphism theorems took up to 9 minutes. Among the more complicated theorems not described in this chapter were proofs that $(a_1A)^{-1}$ is the same set as Aa_1^{-1}, and that for a homomorphism f_1, $\{f_1^{-1}(a_1)\}$ is the same set as a_1K, where K is the kernel of f_1. Other lengthy proofs completed by ADEPT are parts of theorems which cannot be done in entirety by the present program, and will be discussed when inadequacies of ADEPT are considered in a later chapter. Not counting the isomorphism theorems, only 4 proofs took over 2 minutes of computer time; 4 contained 30 or more nodes in the proof tree. (In the intersection of these two classes of difficult theorems were 3 problems.)

The particular problems discussed in this chapter are presented either because they serve as excellent illustrations of various features of ADEPT, or because the problems themselves are of interest, particularly those into which a special insight has been obtained through their use as examples for ADEPT.

Problem V-A

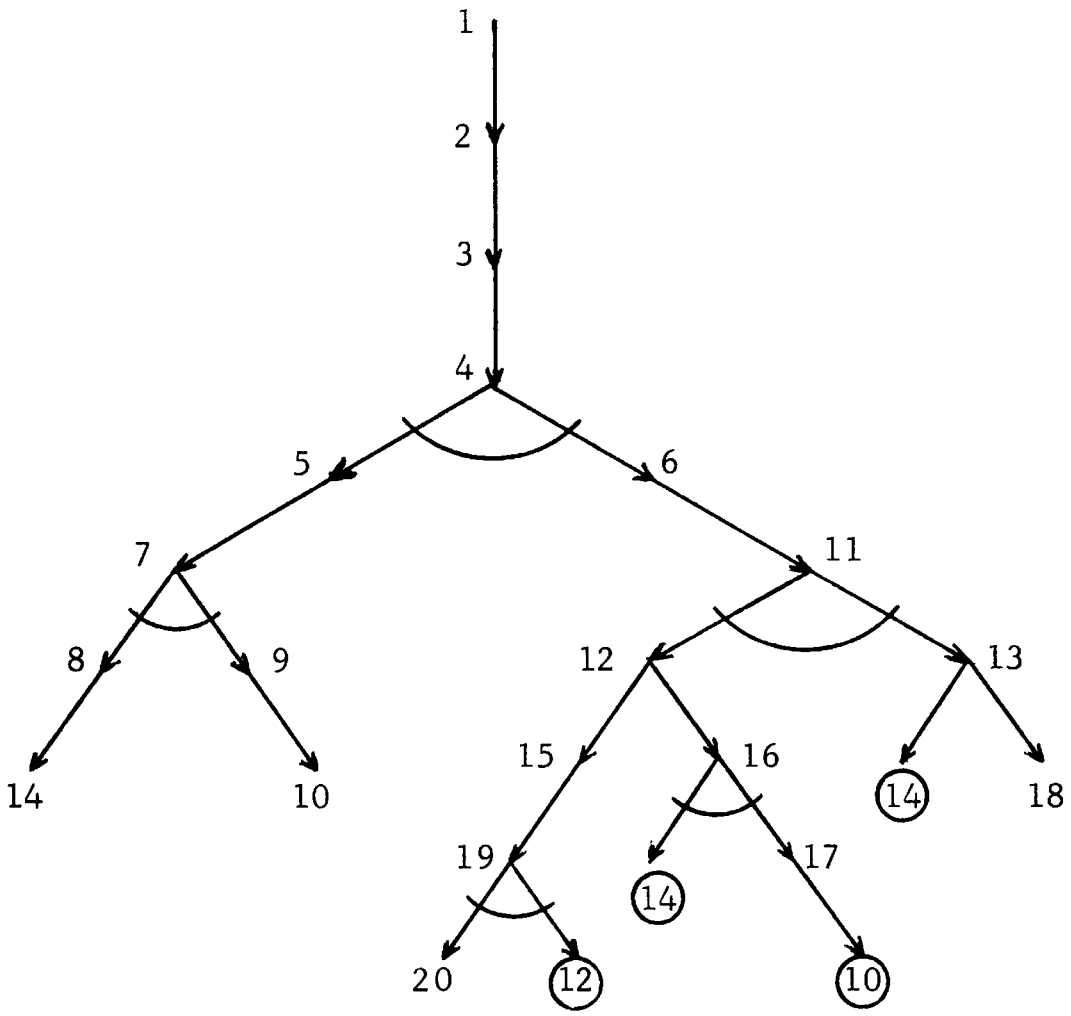
One group of problems presented to ADEPT is concerned with intersections of subgroups:

1) The intersection of two subgroups of a group is itself a subgroup.
2) The intersection of two normal subgroups is itself normal.
3) The intersection of a normal subgroup and an arbitrary subgroup is normal in the arbitrary subgroup.

Consider the last of these from the point of view of stating it formally. There are many possibilities. One is to say in effect — "let I be the intersection of H and J, where H is normal in G and J is a subgroup of G; prove that I is normal in J." This statement may appear to omit some of the content of 3) above (e.g., G is not stated to be a group, and H is not stated to be a subgroup), and indeed it does if one must state explicitly all supporting assumptions. But the facts that H is a subgroup of G and that G is a group are not used explicitly in the proof of this theorem. However, the definition of normality implicitly assumes that the set in question is a subset of a group, because it only makes sense if inverses are defined. (A normal set itself may not necessarily be a subgroup, according to this interpretation.)

These remarks about assumptions have been made just because the proof of 3) can be carried out using the above statement provided that it is assumed that all definitions make sense. This is the way ADEPT has proved this theorem. The proof is now exhibited, following which a discussion of the consequences of explicitly stating the additional justifying hypotheses is presented.

The hypotheses are put on table I: i) I is the intersection of H and J, ii) H is normal in G, and iii) J is a subgroup of G. The first line of table II, the proof tree, is (1) I is normal in J. From the definition of normality



(Circled line numbers indicate references to previously existing lines.)

Figure 3

Proof tree for Problem V-A

is obtained (2) $a_1 \in J \ \& \ a_2 \in I \Rightarrow (a_1a_2)a_1^{-1} \in I$. Assuming an instance of the antecedent results in the addition of iv) $j_1 \in J$ and v) $j_2 \in I$ to table I and of (3) $(j_1j_2)j_1^{-1} \in I$ to the proof tree, where j_1 and j_2 are constants. Status PNT is given to this line. No steps are possible from the hypotheses on table I as it stands. The set constant I is identified by i) and the appropriate instance of the definition of intersection is added to table I: vi) $a_1 \in H \ \& \ a_1 \in J \Rightarrow a_1 \in I$ and vii) $a_1 \in I \Rightarrow a_1 \in H \ \& \ a_1 \in J$. Thus (4) $(j_1j_2)j_1^{-1} \in H \ \& \ (j_1j_2)j_1^{-1} \in J$ goes on table II, and it is split into (5) $(j_1j_2)j_1^{-1} \in H$ and (6) $(j_1j_2)j_1^{-1} \in J$. This application of detachment to (3) causes status PNT to be given to the next line on the proof tree of status REL as soon as all other reductions that can be made from (3) are completed. Since no further steps are possible, the proof continues by examining (5) (since (4) has been disposed of by splitting). Again no progress is possible from table I as it stands. The constant H in (5) causes viii) $a_1 \in G \ \& \ a_2 \in H \Rightarrow (a_1a_2)a_1^{-1} \in H$, an instance of the definition of normality, to be put on table I because of ii). Then the proof tree is augmented by (7) $j_1 \in G \ \& \ j_2 \in H$, which yields (8) $j_1 \in G$ and (9) $j_2 \in H$. Work is now completed on line (5), and line (6) is placed in status PNT. Table I as it stands can cause further progress on table II (all lines from (6) to the end will be considered by SCANW), namely from line (9) is created (10) $j_2 \in I$, which is verified (line v) on table I) and therefore line (9) is also verified. Line (6) must now be examined in detail, and the definition of J is added to table I: ix) $a_1 \in J \Rightarrow a_1 \in G$, x) $a_1 \in J \ \& \ a_2 \in J \Rightarrow a_1a_2 \in J$, xi) $e_G \in J$, and xii) $a_1 \in J \Rightarrow a_1^{-1} \in J$. This leads to the following reduction of (6): (11) $j_1j_2 \in J \ \& \ j_1^{-1} \in J$, which become (12) $j_1j_2 \in J$ and (13) $j_1^{-1} \in J$. This completes consideration of line (6). The next line of status REL is line (8), and it takes on status PNT. Table I as it is yields (14) $j_1 \in J$ as a reduction of (8). Line iv) of table I is verification of this, which results in verification of (7), since the other conjunct of (7) is the already verified line (9). Thus line (5) is verified. Line (12) is the next line to take on status PNT. From vii) on table I is obtained (15) $j_1j_2 \in I$, and then from x), (16) $j_1 \in J \ \& \ j_2 \in J$. As (16) is split, it is seen that $j_1 \in J$ is line (14), already verified, and the

lone reduction of (16) becomes (17) $j_1 \in J$. Next line (13) is scanned, yielding (18) $j_1^{-1} \in I$ and a reference to line (14), and thus line (13) is verified. Attention moves to line (15), which suggests (19) $j_1 j_2 \in H$ & $j_1 j_2 \in J$. This fruitless move is not avoided by ADEPT, and it yields (20) $j_1 j_2 \in H$ and a circular reference to (12). Line (17) is now considered, and vii) on table I gives a reference to line (10), which is verified, thus causing a chain of verification leading through (17), (16), (12), (11), (6), (4), (3), (2), (1), and the proof is complete. The final proof tree is shown in Figure 3. Lines (15), (18), (19), and (20) were unnecessary.

Now suppose that the hypothesis that H is a subgroup of G had been stated explicitly. Line (5) would then have two reductions — the one due to the normality of H (line (7) above, which is split into (8) and (9)), and another due to the hypothesis that H is a subgroup — namely (10') $j_1 j_2 \in H$ & $j_1^{-1} \in H$, which is a useless step. (10') immediately is split into (11') and (12'). In fact, not only is extra work generated, but ADEPT, with its one-pass organization, is unable to prove the theorem at all. The extra hypothesis confounds the algorithm as follows:

After line (5) is processed, line (6) is put into status PNT due to the two applications of detachment to (5). Before the terms and constants of (6) are examined, a scan is done, and line (8) is seen to generate the reduction (13') $j_1 \in H$ by application of detachment using the conjunct of the definition of H as a subgroup which states that H is a subset of G. Line (8) is therefore marked unworthy of further processing by putting it in status REL1. After further lines are generated by the scan, the constants of line (6) are finally examined, and the instance of the definition of subgroup due to the fact that J is a subgroup is put on table I. This suffices to process line (6), as in the proof just presented. But the next line of status REL is no longer line (8), and the assignment of status PNT is made to a later line on the proof tree, thus providing that line (8) never can be scanned by the axioms saying "J is a subgroup". The proof above shows that verification of line (8) is a necessary part of the proof, and that it is verified because J is a subgroup

of G. ADEPT has passed over this crucial line, and therefore can not produce a proof of this version of the theorem.

For the problem of proving that the intersection of two normal subgroups is normal, the introduction of explicit but unnecessary hypotheses (namely that the two normal sets are in fact subgroups) causes ADEPT to do extra work but does not cause the program to be unable to supply a proof. The ADEPT algorithm is unaffected by the manner of statement of the problem that the intersection of two normal subgroups of a group is again a subgroup of that group. As a matter of fact, this proof is done without a single unnecessary line; i.e., all lines on the proof tree contribute to the verification of the head of the tree.

Problem V-B

The second problem to be examined in detail is a rather simple one, though, like the previous example, it cannot be proved by ADEPT if given an unnecessary hypothesis. The proof will be discussed, and a reproduction of the actual output from ADEPT is provided. (Note that table I is not printed out in serial order, and that the symbols VERA2 and A2 are used for the statuses VERST and ST.) Also to be noted in this example is a detail involving the implementation of the simultaneous performance of related substitutions.

Theorem: A subset H of an abelian group G is normal.

Proof: It suffices to show that if $a_1 \in G$ and $a_2 \in H$, then $(a_1 a_2)a_1^{-1} \in H$. Assuming an instance of the antecedent, it remains to prove the corresponding instance of the consequent. In dealing with a particular instance, ADEPT has to generate new symbols internally, which are specified to be constants. The symbols G04088 and G04089 seen in the output are the symbols created by ADEPT for this purpose. For the following discussion, A3 and A4 will be used instead. Thus line (3) would appear as (MEMBER (*PROD (*PROD A3 A4 G) (*INVERSE A3 G) G) H), or in mathematical notation, $(a_3 a_4)a_3^{-1} \in H$. A check of the terms of this line causes no action, since MEMBER has a null sufficient condition and definition, *PROD has a null definition, and there is no definition of *INVERSE in ADEPT's table. (The latter is effectively compiled into PUTON2 by means of "built-in" axioms, as described in Chapter IV.) Examination of the constants of (3) yields two set constants G and H which are defined by three statements on table I. Therefore instances of the corresponding definitions appear on table I, as indicated below (using mathematical notation):

vi) $a_1 \in G \;\&\; a_2 \in G \Rightarrow a_1 a_2 = a_2 a_1$ (from ii), (ABELIAN G G))

vii) $a_1 \in G \;\&\; a_2 \in G \Rightarrow a_1 a_2 \in G$

viii) G is associative

ix) $e_G \in G$

(vii)–(ix): (from iii), (GROUP G))

```
(and (subset h g) (and (abelian g g) (group g)))
nil
(normal h g g)



(((1 . 1) SUBSET H G) ((2 . 2) ABELIAN G G) ((3 . 3) GROUP
G) ((4 . 4) MEMBER G04088 G) ((5 . 4) MEMBER G04089 H) ((9 .
10) MEMBER (*IDENTITY G) G) ((12 . 9) MEMBER (*PROD G04088
G04089 G) G) ((13 . 9) MEMBER (*INVERSE G04088 G) G) ((15 .
9) MEMBER G04089 G))
(((6 . 9) IMPLIES (AND (MEMBER A1 G) (MEMBER A2 G)) (EQUAL
(*PROD A1 A2 G) (*PROD A2 A1 G))) ((7 . 10) IMPLIES (AND (MEMBER
A1 G) (MEMBER A2 G)) (MEMBER (*PROD A1 A2 G) G)) ((8 . 10)
ASSOC G) ((10 . 10) IMPLIES (MEMBER A1 G) (MEMBER (*INVERSE
A1 G) G)) ((11 . 11) IMPLIES (MEMBER A1 H) (MEMBER A1 G)) (
(14 . 9) EQUAL (*PROD (*PROD G04088 G04089 G) (*INVERSE G04088
G) G) (*PROD (*INVERSE G04088 G) (*PROD G04088 G04089 G) G))
((16 . 9) EQUAL (*PROD G04088 G04089 G) (*PROD G04089 G04088
G)))
(((VER (HEAD) (2 NONE) 1 4) NORMAL H G G) ((VER (1) (3 NONE)
2 NILL) IMPLIES (AND (MEMBER A1 G) (MEMBER A2 H)) (MEMBER (
*PROD (*PROD A1 A2 G) (*INVERSE A1 G) G) H)))
(((VER (2) (16 10 NONE) 3 12) MEMBER (*PROD (*PROD G04088 G04089
G) (*INVERSE G04088 G) G) H) ((VERA2 (HEAD) (5 NONE) 4 12 6)
AND (MEMBER (*PROD G04088 G04089 G) G) (MEMBER (*INVERSE G04088
G) G)) ((VERA2 (4) (11 7 NONE) 5 12) MEMBER (*PROD G04088 G04089
G) G) ((VERA2 (4) (NONE) 6 12) MEMBER (*INVERSE G04088 G) G)
((VERA2 (5 HEAD) (9 NONE) 7 12 6) AND (MEMBER G04088 G) (MEMBER
G04089 G)) ((VERA2 (7) (NONE) 8 NILL) MEMBER G04088 G) ((VERA2
(7) (12 NONE) 9 12) MEMBER G04089 G) ((IRR (3) (NONE) 10 12)
NILL MEMBER (*PROD G04088 (*PROD G04089 (*INVERSE G04088 G)
G) G) H) ((IRR (5) (NONE) 11 NILL) NILL MEMBER (*PROD G04088
G04089 G) H) ((VERA2 (9) (NONE) 12 NILL) MEMBER G04089 H) (
(A2 (HEAD) (14 NONE) 13 NILL 6) AND (MEMBER G04088 G) (MEMBER
(*PROD G04089 (*INVERSE G04088 G) G) G)) ((A2 (13) (NONE) 14
NILL) MEMBER (*PROD G04089 (*INVERSE G04088 G) G) G) ((VERA2
(HEAD) (NONE) 15 NILL 6) AND (MEMBER G04089 G) (MEMBER (*INVERSE
G04088 G) G)) ((VER (3) (NONE) 16 NILL) MEMBER (*PROD (*PROD
G04089 G04088 G) (*INVERSE G04088 G) G) H))
QUIT,
R 28.166+1.783
```

Figure 4

Actual output from ADEPT

x) $a_1 \epsilon\ G \Rightarrow a_1^{-1} \epsilon\ G$ (from iii), (GROUP G))

xi) $a_1 \epsilon\ H \Rightarrow a_1 \epsilon\ G$ (from i), (SUBSET H G))

(Line iv) is $a_3 \epsilon\ G$ and line v) is $a_4 \epsilon\ H$). Associativity is expressed by the statement (ASSOC G), allowing the use of a special matching subroutine, as opposed to an explicit line

viii') $a_1 \epsilon\ G\ \&\ (a_2 \epsilon\ G\ \&\ a_3 \epsilon\ G) \Rightarrow (a_1 a_2)a_3 = a_1(a_2 a_3)$.

SCANW now causes the following lines to be put on the proof tree as two subordinate trees:

(4) $a_3 a_4 \epsilon\ G\ \&\ a_3^{-1} \epsilon\ G$

(5) $a_3 a_4 \epsilon\ G$

(6) $a_3^{-1} \epsilon\ G$

(7) $a_3 \epsilon\ G\ \&\ a_4 \epsilon\ G$

(8) $a_3 \epsilon\ G$ (immediately verified)

(9) $a_4 \epsilon\ G$

Verification of (7) or (4) would allow the consequent of vi), the abelian axiom, to cause a substitution in line (3).

From line viii), (10) $a_3(a_4 a_3^{-1}) \epsilon\ H$ is obtained as a reduction of (3). Line (3) has not been involved in a use of detachment, and no verification of it has occurred, so the scan proceeds to the rest of the proof tree (through line (10)). This causes: from (5) because of vii), a reference to (7); from (5) because of xi), (11) $a_3 a_4 \epsilon\ H$; from (6) because of x), verification due to line (8); from (9) due to xi), (12) $a_4 \epsilon\ H$. Since line (12) is line v) of table I, it is immediately verified, thus completing verification of lines (4) through (9), and rendering line (11) irrelevant.

The scan continues, yielding two new subtrees because of vi) and (10):

(13) $a_3 \epsilon\ G\ \&\ a_4 a_3^{-1} \epsilon\ G$

(14) $a_4 a_3^{-1} \epsilon\ G$ ($a_3 \epsilon\ G$ being line (8))

(15) $a_4 \epsilon\ G\ \&\ a_3^{-1} \epsilon\ G$, which is immediately verified since both its conjuncts have been verified.

This completes the scan of the proof tree through line (10) as was indicated, and since new lines have been added to the proof tree, the scan of the

whole tree (lines (3) - (15)) is redone. As shown in Diagram II, no line on table I is used to scan any lines on table II that it already has scanned unless the line is an implication which is now augmented by a verified head of a subordinate tree it suggested. This is the case with line vi), and since line (4) is verified, table I is augmented by xii) $a_3a_4 \in G$, xiii) $a_3^{-1} \in G$, and xiv) $(a_3a_4)a_3^{-1} = a_3^{-1}(a_3a_4)$. Then, analogously, xv) $a_4 \in G$ and xvi) $a_3a_4 = a_4a_3$ are obtained from line (7).

Scanning line (3) with these new lines, two substitutions are found, and since lines xiv) and xvi) have arisen due to the same table I entry (line vi)) and thus are in the same logical class, these substitutions are considered to be related. However, they conflict, and cannot be performed together, so they are done one at a time. As it happens, the substitution suggested by xvi) is performed first, yielding (16) $(a_4a_3)a_3^{-1} \in H$. As described in the last chapter, this is immediately collapsed into $a_4 \in H$, which is already verified as line (12). This suffices to complete the proof.

If line i) of table I had been (SUBGROUP H G), ADEPT would have gone off on a dead end, thinking line (3) to be processed after generation by detachment of the reduction $a_3a_4 \in H$ & $a_3^{-1} \in H$. Efficient means of avoiding such difficulties (i.e., without backtracking) are clearly of great interest, and this subject will be approached in a later chapter.

Problem V-C

Sometimes ADEPT produces a very inefficient proof even when no unnecessary hypotheses have been explicitly stated. This is the case for the following theorem: A conjugate D of a subgroup H of a group G is itself a subgroup of G. (A conjugate of a subgroup H is specified by a member a_1 of G, and is the set such that $a_2 \in D \iff (\exists a_3)[a_3 \in H \ \& \ a_2 = (a_1 a_3)a_1^{-1}]$; i.e., $D = a_1 H a_1^{-1}$.) The proof given by ADEPT for this theorem is 73 lines long, of which 49 are due to "false leads." It is easy to see how this happens by observing the development of only a small section of the proof.

To prove D a subgroup, it is necessary to show closure under composition, so assuming $d_2 \in D$ and $d_3 \in D$, $d_2 d_3 \in D$ must be shown; i.e., $(\exists a_3)[a_3 \in H \ \& \ d_2 d_3 = (a_1 a_3)a_1^{-1}]$ is to be proved. SOLVEX quickly reduces the problem to showing $a_1^{-1}((d_2 d_3)(a_1^{-1})^{-1}) \in H$, which is immediately changed to $a_1^{-1}((d_2 d_3)a_1) \in H$ (*).

The desired continuation is to observe that the membership of d_2 and d_3 in D allows substitution of $(a_1 h_1)a_1^{-1}$ and $(a_1 h_2)a_1^{-1}$ for d_2 and d_3 respectively in (*), where h_1 and h_2 are then known to be in H. The two substitutions are related, and will be performed simultaneously, yielding the following reduction of (*): $a_1^{-1}((((a_1 h_1)a_1^{-1})((a_1 h_2)a_1^{-1}))a_1) \in H$, which immediately will be collapsed into $h_1 h_2 \in H$. This soon will be verified inasmuch as H is a subgroup. But, from (*) are derived 3 other reductions, each of which leads to much extra work. Two of these arise because of associativity, and the third is a direct application of the closure-under-products conjunct of the definition of the subgroup H to line (*). All of these entries are on table I by this time, and so there is no selectivity involved. A similar situation arises when proving the closure of D under inverses. Perhaps the greatest difficulty in theorem-proving is "knowing" when a proof is on the right track. Once again, the reader is referred to the later chapters.

Problem V-D

A lesson in the use of ADEPT may be learned from the following example.

The commutator of two members a_1 and a_2 of a group G, denoted in mathematical notation by $[a_1,a_2]$, is defined by $[a_1,a_2] = ((a_1^{-1}a_2^{-1})a_1)a_2$. This concept can, of course, be defined for ADEPT, and identities concerning commutators can be proved. ADEPT is not primarily designed for manipulation of identities, and all but the simplest of these requires a large amount of computer time, though there are some fairly obvious heuristics which could be applied to this very special problem area to remedy this. Such digressions are not relevant to the ADEPT project, but insights from one particular simple identity problem happen to be valuable in general. The problem is:

$[a_1,a_2]^{-1} = [a_2,a_1]$.

Although either the notation (COMMUTATOR A3 A1 A2 G) or (*COMMUTATOR A1 A2 G) may be employed, assume the latter is used. The problem then may be stated in many ways. One is to give ADEPT the conclusion (EQUAL (*INVERSE (*COMMUTATOR A1 A2 G) G) (*COMMUTATOR A2 A1 G)). Remembering that no substitutions are made in the second argument of an equality on the proof tree, it is seen that this will produce a fairly lengthy chain of substitutions, resulting in a correct proof, but a proof made quite long due to the application of the associativity axiom. No improvement is obtained using FEQUAL. For this problem there is a way out, however. A sufficient condition, known to ADEPT, for INVERSE, is given by "a_2 is the inverse of a_1 if $a_2a_1 = e$". This can be used if the usual asterisked notation for inverse is not employed; i.e., if the conclusion of the problem is put in the form: (INVERSE (*COMMUTATOR A2 A1 G) (*COMMUTATOR A1 A2 G) G). Doing this works very well, simply because related substitutions are made simultaneously, so that the reduction of the application of the sufficient condition is $(((a_2^{-1}a_1^{-1})a_2)a_1)$ $(((a_1^{-1}a_2^{-1})a_1)a_2) = e$, which is immediately collapsed into $e = e$, a line

which is verified as it is put on the proof tree. This is a very striking example of the value of doing related substitutions simultaneously and performing certain operations on a line, such as this "collapsing", at the time of its placement on the proof tree.

The point of all this is that different ways of doing problems are often not too similar in their demands upon the resources of computer time and memory. Even if ADEPT were much more highly developed than it is at present, including possession of the ability to choose the "correct" path of the proof tree more often, it well may not be able to take the statement of a problem and transform it into the most advantageous form. ADEPT should be able to solve the problem in any form, sooner or later, but the user can be asked to utilize some ingenuity in setting up the form of the hypotheses, conclusions, and definitions. ADEPT is flexible enough to allow significant experimentation along these lines, perhaps to the end that future programs will be able to take this burden from the user.

Problem V-E

The following is a problem that has been considered by other theorem-proving programs: a group, every element of which is of order 2, is abelian. ADEPT has no provision for numbers, but for an element a_1 to be of order 2 simply means that it must satisfy the equation $a_1a_1 = e$. One would like to give this problem to ADEPT and have it come up with a proof. But, faced with proving $g_1g_2 = g_2g_1$, the program has no idea how to proceed. None of the axioms can be applied to the statement $g_1g_2 = g_2g_1$. Now it turns out that a simple proof of this theorem, requiring no "construction," follows from the consequence of $a_1a_1 = e$ that $a_1 = a_1^{-1}$. The proof proceeds: $g_1g_2 = g_1^{-1}g_2^{-1} = (g_2g_1)^{-1} = g_2g_1$. ADEPT proves this theorem given the hypothesis $a_1 = a_1^{-1}$. It does so with very little wasted effort, due to the restriction on matching simple variables and the immediate simplification of expression such as $(g_1^{-1})^{-1}$.

Of course, this is hardly fair, for in order to have ADEPT prove this theorem, the user must know how the proof proceeds! Clearly ADEPT has to be more clever, though a user who was very familiar with the program might realize which of the two ways of stating the "order 2" hypothesis was preferable (that is, if he were aware of both alternatives). One way to approach this difficulty would be to program into ADEPT a feature whereby insertion of the fact $a_1a_1 = e$ on table I causes the consequence $a_1 = a_1^{-1}$ to appear. This would have to be programmed carefully, so that very little time would be spent checking all lines being put on table I to see if they are of this form. Such a feature would properly be called a heuristic.

Another approach would be to make explicit use of a lemma, $a_1a_1 = e \Rightarrow a_1 = a_1^{-1}$. Under which conditions this lemma would be put on table I is a difficult problem. In addition, even if this lemma were not brought onto table I to "confuse" proofs of other problems, its use in this problem causes

a proof of twice the length, requiring four times the time of the original proof with the simple hypothesis $a_1 = a_1^{-1}$.

The empirical approach to theorem-proving demands the incorporation of such lemmas or heuristics as the one described above. There are certainly direct analogues to the inclusion of such facts in ADEPT in the learning process of a student of group theory. This is in contrast to specification of a "complete" proof procedure (e.g., Herbrand's), which has a different orientation than that of a student. It is known that complete procedures are very prone to fatal combinatorial explosion. The question is: Can an empirical approach handle enough special cases efficiently to surpass the other approach?

Problem V-F

One of the first problems considered in conjunction with ADEPT was one of the most troublesome: The kernel of a homomorphism f_1 reduces to the identity if and only if f_1 is one-to-one. This theorem can be divided into two parts, and, defining a unitset to be a set with a single element, details of this problem will now be considered.

Let f_1 be a homomorphism from G to H, K its kernel, E the unitset containing the identity of G, and first assume that K is a subset of E; i.e., K reduces to the identity. To be shown is that f_1 is one-to-one. A sufficient condition for f_1 to be one-to-one is that if $f_1(a_1) = f_1(a_2)$, then $a_1 = a_2$. The usual proof proceeds by contradiction; if $f_1(a_3) = f_1(a_4)$ does not imply $a_3 = a_4$ for some a_3 and a_4, then $f_1(a_3a_4^{-1}) = f_1(a_3)f_1(a_4)^{-1}$ (since f_1 is homomorphic) $= e_H$, so $a_3a_4^{-1} \in K$. However, $a_3a_4^{-1} \neq e_G$ since $a_3 \neq a_4$, contradicting the assumption that K reduces to e_G.

ADEPT has no provision for proofs by contradiction, but a reformulation of the sufficient condition is all that is needed to allow ADEPT to solve this problem: f_1 is one-to-one if $f_1(a_1)f_1(a_2)^{-1} = e_H$ implies $a_1a_2^{-1} = e_G$. This is clearly equivalent, and the proof proceeds straightforwardly: $a_3a_4^{-1} = e_G$ if $a_3a_4^{-1} \in E$, if $a_3a_4^{-1} \in K$, if $f_1(a_3a_4^{-1}) = e_H$, but $f_1(a_3a_4^{-1}) = f_1(a_3)f_1(a_4^{-1}) = f_1(a_3)f_1(a_4)^{-1} = e_H$. Thus the formulation of the sufficient condition is important, and it is reasonable to expect a user of ADEPT to supply appropriate entries to its tables, just as in Problem V-D it was argued that the user should be expected to employ ingenuity in choosing the form of statement of a theorem.

For the converse, it is the definition of one-to-one that is important, and indeed this half of the theorem is more treacherous. Here, assuming f_1 is one-to-one, it is desired to show that K is contained in E. So assume $k_1 \in K$ and attempt to prove $k_1 \in E$; i.e., $k_1 = e_G$. If the definition of one-to-one

used is that if $f_1(a_1) = f_1(a_2)$, then $a_1 = a_2$, an infinite loop is generated: $k_1 = e_G$ if $f_1(k_1) = f_1(e_G)$ if $f_1(f_1(k_1)) = f_1(f_1(e_G))$, etc. This loop could be stopped artificially, but that seems to be an unnecessary extra check to add to the program. Alternatively, an effort could be made to make sure that the argument of a function was in its domain, but this would not help in the case G = H. If the definition of one-to-one is: $f_1(a_1)f_1(a_2)^{-1} = e_H$, ==> $a_1a_2^{-1} = e_G$, ADEPT will have no "ideas" as to how to proceed. But, the connective FEQUAL is available, and the definition of one-to-one can very naturally be phrased $f_1(a_1) =_f f_1(a_2)$ ==> $a_1 = a_2$, and the proof proceeds: $k_1 = e_G$ if $f_1(k_1) =_f f_1(e_G)$. Evaluating the left-hand side: $f_1(k_1) = e_H$ since $k_1 \in K$. Evaluating the right-hand side: $f_1(e_G) = e_H$ since f_1 is homomorphic. That completes the proof of this theorem, which is another example where the forms of definitions, etc. are critical.

Problem V-G

Among the proofs of isomorphisms produced by ISOLVE (see Chapter III), none contain any surprises. The proofs have been of isomorphisms of a trivial to a relatively complex nature. The simplest required ISOLVE to go through the motions of proving a group G isomorphic to itself. As a related problem, GENFCN may be overridden, and the relation $f_1(a_1) = (g_1a_1)g_1^{-1}$ for some $g_1 \in G$ inserted, and this will be shown to be an isomorphism. Similarly, for G abelian, $f_1(a_1) = a_1^{-1}$ can be shown to be isomorphic by ISOLVE.

The most complicated isomorphism that has been done by ISOLVE is to prove that G/H is isomorphic to (G/K)/(H/K). This statement only makes sense for the case that K and H are normal subgroups of G, and ADEPT is able to prove the justifying fact that, under these hypotheses, H/K is a normal subgroup of G/K. To prove the isomorphism, the assumptions of the normality of K and H need not be stated explicitly, as they play no further role in the proof. The entire proof of isomorphism, including the generation of a map from G/H to (G/K)/(H/K) by GENFCN, took 7½ minutes of computer time and 40 entries on the proof tree. Of these, 7 were used in the proof that the relation is well-defined, 18 in the proof that it is homomorphic, 12 in the proof that it is onto, and 3 to prove that it is one-to-one. 2 lines of the proof that it is well-defined and 6 lines of the proof that it is homomorphic were irrelevant; i.e., useless branches. (Incidentally, if GENFCN is asked to provide a relation from (G/K)/(H/K) to G/H, the corresponding proof of isomorphism takes 9 minutes of computer time and 44 lines.) Without the use of the MODEL heuristic, the same proof would have been generated, but with an additional 30 to 40 irrelevant lines at a cost of 3 to 4 extra minutes of computer time.

Other isomorphisms established by ISOLVE include that of G and K when f_1 is an isomorphism from G to H and f_2 is an isomorphism from H to K; that of G with $G/\{e_G\}$; that of $G/(\text{Ker } f_1)$ with $f_1(G)$ for a homomorphism f_1; and that of

$G/f_1^{-1}(K)$ with H/K for an epimorphism f_1 from G to H (where K is a normal subgroup of H). ADEPT is able to prove all justifying theorems to show that these problems make sense — for instance, in the last example, that $f_1^{-1}(K)$ is indeed a normal subgroup of G. ISOLVE does not automatically go through these justifying theorems; it assumes the problem to be well-defined. But ADEPT has been given the associated theorems which establish this, and it has proved them.

ISOLVE may also be used to establish theorems about other relations. For instance, rather than give ADEPT separate theorems, ISOLVE may be run in an attempt to "prove" that the relation (generated by GENFCN) $f_1(a_1) = a_1H$ from G to G/H is an "isomorphism". ISOLVE will produce proofs that the relation is homomorphic and onto, and will, of course, fail to show it to be one-to-one. Another such example is the canonical relation between G/K and $f_1(G)/f_1(K)$ which also is an epimorphism but not an isomorphism, for a general epimorphism f_1. This relation must also be proved to be well-defined, as opposed to the map from G to G/H which is trivially well-defined by its defining equation.

It is interesting to observe that though the term *LCOSET (which stands for "left coset") appears often in the statement on the proof trees of the isomorphisms involving factor groups, its definition, which is not needed in order to establish the isomorphisms, is never put on table I by ADEPT, thus saving a great deal of effort. This is a good illustration of why it is not desirable to go to the tables for instances of definitions of all terms that appear in the course of a proof regardless of whether or not they appear in lines of status PNT.

To conclude this chapter, an insight gained from ISOLVE should be mentioned. The selective introduction of instances of definitions; i.e., the controlled growth of table I at the start of a problem, mentioned in Chapter III, could, of course, be used by ADEPT in each of the 4 sub-theorems handed to it by ISOLVE. This involves, in general, obtaining the definition of each term in the hypotheses of an isomorphism theorem 4 times. It has been found that the class of proofs involved in establishing isomorphisms is not sensitive

to selective growth of table I, and consequently it is more profitable to have ISOLVE call ADEPT with the instances of definitions of all terms in the hypotheses already on table I. Thus ISOLVE can consult the table of definitions once for all 4 sub-theorems. The determination of the relative merits of such trade-offs is one of the types of insights into proving theorems about groups by computer which can easily be obtained by experimentation with ADEPT.

CHAPTER VI

LIMITATIONS OF THE PRESENT PROGRAM

The previous chapter, in addition to illustrating proofs produced by ADEPT, has also served as an introduction into the limitations of the present program. The remainder of this report will be devoted almost entirely to discussions of various types of theorems with which ADEPT cannot cope. These types fall into different classifications. For instance, there are those which ADEPT cannot do simply because it is not broad enough. Such theorems include facts involving the introduction of numbers, say as orders of finite groups. Another such group is theorems requiring statements using double existential quantifiers, which cannot be handled by SOLVEX at present. These theorems are not significantly harder in any sense of the word; they simply require additional subroutines for ADEPT.

Another class of theorems might be called the overspecified class. Such a grouping has no mathematical basis in group theory, but it is a label to be applied to those theorems with extra hypotheses with which ADEPT performs so erratically, as seen in the last chapter. As theorems become more complex, even those with minimal sets of hypotheses will cause similar difficulties for ADEPT. A decision must be made whether or not this can be overcome by heuristics, or whether a completely different approach to theorem-proving will ultimately have to be adopted.

There are also theorems which require new and different methods of proof. In contrast to those which require introduction of additional types of concepts, these theorems are often significantly more difficult. Mechanization of proof by contradiction, use of constructions in proofs, etc., all must be considered. Finally, it is important to ask if the reasoning used in advanced

theorems of group theory is fundamentally different from that used in elementary theorems? Is it the case that it is not a matter of extending ADEPT to be much, much more sophisticated, but rather a matter of implementing an entirely different form of human thought process, though perhaps in conjunction with a more sophisticated ADEPT? The logician is tempted to rely upon the theoretical fact that all theorems of mathematics follow or issue from the basic axioms of the system; the pure mathematician has reservations for he knows that intuition and other rather illogical procedures play a large part in his advanced work. The choice of methods with which to prove an advanced theorem is an excellent example of a step which cannot be done by simple processing of hypotheses, conclusion, and relevant definitions.

All these considerations must be taken into account when discussing the future of theorem-proving by computer, and when discussing the level to which a program like ADEPT for proving one class of theorems should be perfected.

The class of theorems for which ADEPT is best suited should be clear. ADEPT produces proofs which follow directly from given axioms, lemmas, and definitions. To some extent this can be augmented, perhaps by such methods as proof by contradiction. But clearly the approach used in ADEPT is ill-suited to many methods of proof, such as construction of counter-examples, a procedure certainly in the repertoire of any good group theorist.

Before continuing this discussion, it will be profitable to consider specific problems of two kinds. The first kind includes problems that ADEPT either can "almost" prove, or that are closely related in content to problems that ADEPT has proved. The second type is a group of problems of a different nature, involving new concepts and methods, which have been considered in detail with proof by ADEPT in mind.

Problem VI-A

It would certainly appear that ADEPT should be able to prove that the center of a group is a normal subgroup. However, it cannot, and the reasons why are instructive.

Proving that the center is closed under products is done easily, by a version of the following argument: Let C be the center of a group G, and let c_1 and c_2 be members of C. To show $c_1c_2 \in C$ requires proving that $(c_1c_2)g_1 = g_1(c_1c_2)$ for $g_1 \in G$. But $(c_1c_2)g_1 = c_1(c_2g_1) = c_1(g_1c_2)$, since $c_2 \in C$, and $c_1(g_1c_2) = (c_1g_1)c_2 = (g_1c_1)c_2 = g_1(c_1c_2)$, since $c_1 \in C$. There is inefficiency in ADEPT's actual proof, to be sure, but it proceeds straightforwardly.

The next step is to prove closure under inverses. Assuming $c_1 \in C$, $c_1^{-1} \in C$ must be proved, or $c_1^{-1}g_1 = g_1c_1^{-1}$ for any $g_1 \in G$. There is more than one way to do this, but no way that does not require a bit of ingenuity. One method is to use a "construction"; i.e., the reverse of the procedure of simplification. This proof proceeds: $c_1^{-1}g_1 = c_1^{-1}g_1c_1c_1^{-1} = c_1^{-1}c_1g_1c_1^{-1} = g_1c_1^{-1}$, since $c_1 \in C$. This approach requires clever heuristics to introduce the appropriate construction. An even neater proof is as follows: $c_1^{-1}g_1 = (g_1^{-1}c_1)^{-1} = (c_1g_1^{-1})^{-1} = g_1c_1^{-1}$. This proof depends upon the identities $a_1^{-1}a_2 = (a_2^{-1}a_1)^{-1}$ and $a_1a_2^{-1} = (a_2a_1^{-1})^{-1}$. Now ADEPT "knows" the identity $(a_1a_2)^{-1} = a_2^{-1}a_1^{-1}$, as explained in Chapter IV, and this suffices to provide one of the equalities needed for this proof, for $(c_1g_1^{-1})^{-1}$ matches $(a_1a_2)^{-1}$, i.e., is of that form. But $c_1^{-1}g_1$ does not match $a_2^{-1}a_1^{-1}$; such a match requires an explicit intermediate step — $c_1^{-1}g_1 = c_1^{-1}(g_1^{-1})^{-1}$ — to obtain the proper form.

It is not profitable for ADEPT to explicitly check every line against the identity $a_1^{-1}a_2 = (a_2^{-1}a_1)^{-1}$, even though it seems natural to check for instances of the identity $(a_1a_2)^{-1} = a_2^{-1}a_1^{-1}$. Neither is it profitable to use

$a_1 = (a_1^{-1})^{-1}$ as a symmetric equality. (Remember that it is written now using EQUAL, not EQUAL2. This is an example where extra programming is dictated to make possible the reverse substitution, rather than opening the problem up to a proliferation of substitutions by using EQUAL2.) So again heuristics need to be discovered to enable ADEPT to prove this theorem without introducing provisions in the algorithm which would virtually insure large increases in useless effort for all proofs in general.

Another mishap occurs when trying to prove that the center is normal. Assuming $c_1 \in C$ and $g_1 \in G$, the desired conclusion is $(g_1c_1)g_1^{-1} \in C$. The "obvious" step to ADEPT is to proceed as in the first two parts of this theorem, namely to show that $((g_1c_1)g_1^{-1})g_2 = g_2((g_1c_1)g_1^{-1})$ (*) for any $g_2 \in G$, a reduction obtained by detachment, using the definition of center. And the algorithm provides that no other use of the definition of center can then be made with respect to the line $(g_1c_1)g_1^{-1} \in C$ (see Diagram II and the discussion of heuristic B for "progress" in Chapter IV). For most theorems, this restriction of ADEPT is well-justified, for most theorems progress as the first part of this one did. To prove $c_1c_2 \in C$, ADEPT showed $(c_1c_2)g_1 = g_1(c_1c_2)$, and to carry out a parallel branch starting with $c_2c_1 \in C$ is needless proliferation of effort. However, to show normality the simple proof is to proceed $(g_1c_1)g_1^{-1} \in C$ if $(c_1g_1)g_1^{-1}$ $(= c_1) \in C$. Indeed, to prove (*) requires either some kind of planning, a construction, or substitution in both sides of (*), none of which are presently possible in ADEPT.

Thus, though ADEPT has enabled the computer to produce many proofs, emphasis on those types of problems which give it trouble indicates that a radical re-examination of the whole philosophy of ADEPT's design may be necessary. Or perhaps the difficulty is that the proper heuristics have not yet been isolated.

Problem VI-B

There is a second problem on which ADEPT fails that is not dissimilar from the preceding one. Consider the theorem that the map $f_1(a_1) = a_1^{-1}$ from a group G into itself (the map taking every element into its inverse) is homomorphic if and only if G is abelian. One half of this gives ADEPT no trouble; in fact for G abelian, the map in question is isomorphic and ISOLVE proves this.

The converse is not hard, but requires ingenuity of the same type as that needed in the second part of the previous theorem. To show G abelian, ADEPT must establish the identity $g_1g_2 = g_2g_1$ under the hypothesis that f_1 is homomorphic. Completing the proof requires use of a "construction"; in particular, a substitution of the form $a_1 \leftarrow (a_1^{-1})^{-1}$. Note that the execution of such a step would require a message to the simplifying routine associated with PUTON2, insuring that the new line will not be simplified. The proof proceeds: $g_1g_2 = ((g_1g_2)^{-1})^{-1} = f_1(g_1g_2)^{-1} = (f_1(g_1)f_1(g_2))^{-1} = (g_1^{-1}g_2^{-1})^{-1} = ((g_2g_1)^{-1})^{-1} = g_2g_1$. (Close examination of this proof reveals that the hypothesis defining the map f_1 must be written using EQUAL2. The converse of this theorem, showing f_1 to be an isomorphism when G is abelian, requires only EQUAL in the defining equation of the map.)

Further discussion of this theorem would only parallel comments made while discussing the last problem. However, in Chapter VIII some possible means of overcoming difficulties such as these will be suggested.

Problem VI-C

It is a theorem of group theory that $f_1(K)^{-1} = K$ for any subset K of f_1's domain, if f_1 is a monomorphism. The inclusion $K \subset f_1(K)^{-1}$ is trivial and is true for any homomorphism f_1. ADEPT proves it with ease. The converse is much harder, and appears to demand an approach not possible with ADEPT. At least, the most obvious proof to the author is one which is ill-suited to ADEPT's algorithm. It seems to require a controlled "working forward" approach which cannot be supplied by the "lemma-proving" procedure which is made possible by lines of status ST on table II. Briefly, the proof proceeds as follows:

Assume $a_1 \in f_1(K)^{-1}$. Then $f_1(a_1) \in f_1(K)$, which, by the definition of image, means that $f_1(a_1) = f_1(a_2)$ for some $a_2 \in K$. But f_1 is one-to-one, so $a_2 = a_1$ and therefore $a_1 \in K$, which was to be shown.

Now consider the problem as ADEPT sees it — given the assumptions and assuming $a_1 \in f_1(K)^{-1}$, the line to be proved is $a_1 \in K$. But K is an arbitrary subset, and in addition no substitutions are possible for a_1, so ADEPT is unable to continue. From the point of view of working backwards, the above proof depends on the implication $(\exists a_2)[a_2 \in K \,\&\, a_1 = a_2] \Longrightarrow a_1 \in K$. However, even if ADEPT "knew" this fact, SOLVEX would attempt to "solve" the line $(\exists a_2)[a_2 \in K \,\&\, a_1 = a_2]$ for a_2, and would come up with the "reduction" $a_1 \in K$, which is no help at all. Thus determination of a candidate for this variable a_2 must proceed by more devious methods. The fact that the definition of $f_1(K)^{-1}$ causes $a_1 \in f_1(K)^{-1}$ to imply an identity involving $f(a_1)$ as opposed to a_1 may be able to be used as the "tip-off" that the usual procedures of ADEPT will not suffice for this proof.

Problem VI-D

The last problem which will be discussed to illustrate limitations of ADEPT for proving theorems of the type it is best suited to prove is the necessary and sufficient condition for equality of two (left) cosets a_1H and a_2H: $a_1H = a_2H \iff a_2^{-1} \in H$. This theorem is an example of how everything can go wrong!

First, consider proving $a_2^{-1}a_1 \in H$ from the equality of the cosets. ADEPT has no "ideas" except to attempt to show a_1 and a_2 to be members of a_1H or a_2H, which is suggested due to the form of the definition of left coset, for if any of these lemmas could be established, a substitution would be possible in the original line. Now a_1 and a_2 are general members of a group G of which H, a_1H, and a_2H are subsets. Therefore the MODEL heuristic will not allow these candidates for heads of subordinate trees to appear on table II. Fortunately, the user has the option of not using the MODEL lattice (though he may not know when not to use it), and at least it is possible to get ADEPT to proceed with this proof.

Since a_1H and a_2H are the same set according to the hypotheses, each of these four heads of subordinate trees is a reduction of one of the other three. For example, to show $a_1 \in a_2H$ it suffices to show $a_1 \in a_1H$. The fact that $a_3 \in a_3H$ for any a_3 is a simple one, and ADEPT can prove it quickly if given it as an explicit problem. However, as a line of status ST, it cannot be proved, for the proof that $a_3 \in a_3H$ requires "solving" by SOLVEX the line $(\exists a_4)[a_4 \in H \ \& \ a_3 = a_3a_4]$. As shown in Diagram I, only lines of status PNT are ever processed by SOLVEX, and a line of status ST can never take on status PNT. Thus the present version of ADEPT is stymied.

One possibility, which would not cause great amounts of additional effort per proof, would be to try to "solve" every such line put on table II (i.e.,

always trying a call to SLVX in Diagram IV). If the line were not immediately solved, it would not be scanned as described in Chapter III unless it were of status PNT. The example in this theorem can be solved immediately, yielding $a_3^{-1}a_3 \in H$ which quickly reduces to $e_G \in H$, which could be verified from part of the definition of subgroup. But again trouble develops, for if H were explicitly assumed to be a subgroup, the original conclusion would be reduced to showing $a_2^{-1} \in H$ & $a_1 \in H$, which is certainly a fruitless attempt at a solution. Furthermore, this reduction is derived by detachment, and thus $a_2^{-1}a_1 \in H$ would be removed from further consideration, and by the time the status ST lines were verified, the substitutions in the original conclusion could no longer be made, since the pointer would have been moved down (i.e., status PNT would have been assigned to some subsequent line of the proof tree).

It has already been observed in Chapter III that the design of ADEPT, and in particular, the restricted processing given to lemmas, is not oriented to proofs which must go back many "levels" to original definitions. In this spirit one would suggest that the user include with the specific hypotheses of this theorem the lemma $a_3 \in a_3H$. Even with this approach, the user can run afoul! The use of asterisked terms happens to be critical in this instance. Assuming that the user inputs the lemma in the form (MEMBER A3 (*LCOSET A3 H G)) (where A3 is a variable), the proof will be completed easily. Without the asterisked term, the user must state the lemma as a translation of "let A be a left coset a_3H; then $a_3 \in A$", and this would cause an undesirable application of detachment to the original conclusion. Since $a_3 \in A$ matches $a_2^{-1}a_1 \in H$, the proof would be "reduced" to showing that H is the coset $(a_2^{-1}a_1)H$. This inference is completely valid, and the new line does happen to be provable, but only with great effort. Thus there is a definite need for the use of the asterisked term in this problem.†

The converse of this theorem is really a conjunction, namely to prove that $a_1H \subset a_2H$ and $a_2H \subset a_1H$ if $a_2^{-1}a_1 \in H$. This proceeds normally, and requires explicit assumption that H is a subgroup. One inclusion parallels the other

in its proof, though ADEPT has no way of detecting this and profiting by the observation. The parallel is not quite complete; proving the inclusion $a_2H \subset a_1H$ comes down to observing $a_1^{-1}a_2 \in H$, which, while implied by $a_2^{-1}a_1 \in H$, is not derivable by ADEPT. This situation is analogous to the one in the first problem described in this chapter, and no more need be said here.

This discussion of this last theorem may seem to be lengthy and to place undue emphasis on the limitations of ADEPT. However, despite the inept performance just outlined, ADEPT did manage to prove many theorems! But all heuristics break down sooner or later, and a clear discussion of the implications of ADEPT's organization was certainly in order. Shortly, discussion will resume on the worth, limitations, and future of ADEPT, including possible corrective modifications which could be made, but first, problems of another class should be discussed — those with new features to which ADEPT could be adapted. After all, future work must take into account more kinds of theorems and harder theorems, as well as theorems very similar to the ones already discussed.

---

† Note that a more general statement of the lemma, such as $e_G \in B \Rightarrow a_3 \in a_3B$, could lead to still another undesirable effect if it were not stated using the asterisked term *LCOSET. If the table I entry were (IMPLIES (AND (MEMBER (*IDENTITY G) B) (LCOSET A A3 B G)) (MEMBER A3 A)), then any occurrence of detachment using this implication would give rise to a table II entry with a variable, namely B. While variables in table II entries are not forbidden, they can lead to an entry which is difficult to prove or which has no interpretation. Here a proof tree line $a_4 \in H$ could be reduced to a conjunction, one conjunct of which is $e_G \in B$, for a completely unspecified variable B. Thus the user must beware of implications $P \Rightarrow Q$ on table I which have the property that if a table II entry Q' leads to a reduction P' using detachment, then P' contains (free) variable; i.e., in creating the instance P' using the information obtained in finding the match of Q' and Q, constants were not substituted for all of the variable symbols of P. This situation can be avoided by making use of the well-known equivalence (used in the reduction of statements to prenex normal form): $(\forall a_1)[Pa_1 \Rightarrow Q] \equiv (\exists a_1)Pa_1 \Rightarrow Q$. In the above

example, the lemma could have been stated (IMPLIES (EXISTS B (AND (MEMBER (*IDENTITY G) B) (LCOSET A A3 B G))) (MEMBER A3 A)). With this statement, trouble is averted, though, of course, an extension to SOLVEX would be necessary to actually handle any reductions generated by detachment using it.

CHAPTER VII

EXTENSIONS OF ADEPT

One method of proof which ADEPT does not use is mathematical induction, and it was this method which was considered in most detail as a specific extension of ADEPT. Theorems which are proved naturally by induction are not too common in group theory, even when groups of finite order are considered, but enough were isolated in order to justify this section of the investigation. Most involve terms with a varying number of arguments, such as a finite product or intersection. Consequently, important questions of notation arise. It is simple enough to define a new type of variable or constant to represent integers, and N, N1, ..., N9 are used for the integer symbols, which may be variables or constants. Having done this, how shall the new concepts be represented?

The easiest concept to represent is the iterated product of the same variable — $\overbrace{a_1 \ldots a_1}^{n \text{ terms}}$ or $(a_1)^n$. Using an asterisked term, this may be written (*EXP A1 N G), where G is the set on which the composition is defined. The general iterated product is more difficult. Consider the product $a_1 \ldots a_n$, which is to be denoted by the term *GPROD with some appropriate set of arguments which will include G, the set with the composition. Any notation used for this must be flexible enough to allow easy statement of such products as $f_1(a_1) \ldots f_1(a_n)$. It must also be easy to express the fact that if n is 1 in either product, the generalized product collapses to a single object. But this object (a_1 or $f_1(a_1)$ in the examples given) is <u>subscripted</u>. It is important to notice that the subscripts of the letter "a" in the following two expressions are not used in the same way: (1) $a_1 \in G \;\&\; a_2 \in G \Rightarrow a_1 a_2 \in G$;

(2) $f_1(a_1 \ldots a_n) = f_1(a_1) \ldots f_1(a_n)$. In (1), the subscripted letters are used as symbols for simple variables. The particular integers used as the subscripts have no significance. In (2), the subscripts are used to specify an ordering of the variables. Indeed, in (2), the unsubscripted letter is a meta-symbol denoting any variable/constant symbol, and the subscript yields information on the order and number of these symbols.

Retaining the spirit of Polish notation, the term *SUB is introduced, to be used whenever a subscript is used to specify ordering, etc. A variable symbol can easily play the part of the meta-symbol, and a tentative representation for $a_1 \ldots a_n$ is (*GPROD (*SUB A9 1) (*SUB A9 N) G).

This is not satisfactory, and in order to see this, consider the equalities $a_1 \ldots a_{n'} = (a_1 \ldots a_n)a_{n'}$ and $f_1(a_1) \ldots f_1(a_{n'}) = [f_1(a_1) \ldots f_1(a_n)]f_1(a_{n'})$. (n' is the successor of n, ordinarily n + 1.) These are both instances of one general fact, which should be expressible by one statement. However, in the corresponding representations for these identities, the occurrences of (*SUCCESSOR N) are at different levels of the expression, and the matching routine will not suffice. Stating the situation in another way, it is impossible for ADEPT to have a schema of the form P(n) = Q(n) where n occurs at arbitrary levels in the syntax of P.

One possible notation which will at least suffice, though it is cumbersome, is to include as arguments of *GPROD:

i) the subexpression of ii) which is the subscripted subexpression of the general element;

ii) a symbol acting as a meta-symbol for the general element of the product;

iii) the number of elements in the product;

iv) the set upon which the composition is defined.

Similarly, a single subscripted object is written using as arguments of *SUB:

i) same as i) for *GPROD;

ii) same as ii) for *GPROD;

iii) the value of the subscript.

The first argument mentioned is present to allow for a distinction between $f_1(a_n)$ and $[f_1(a)]_n$. Thus $f_1(a_1)\ldots f_1(a_{n'}) = [f_1(a_1)\ldots f_1(a_n)]f_1(a_{n'})$ can be written: (EQUAL (*GPROD A1 (F1 G H A1) (*SUCCESSOR N) H) (*PROD (*GPROD A1 (F1 G H A1) N H) (*SUB A1 (F1 G H A1) (*SUCCESSOR N)) H)). This is an instance of the general statement (EQUAL (*GPROD A8 A9 (*SUCCESSOR N9) A) (*PROD (*GPROD A8 A9 N9 A) (*SUB A8 A9 (*SUCCESSOR N9)) A)), where A8, A9, N9, and A are (free) variables. It is not necessary to indicate in some way that A8 must match a subexpression of whatever expression A9 matches, for this will automatically be the case in any well-formed use of *GPROD.

Together with the preceding general statement, the identity (EQUAL (*GPROD A8 A9 1 A) (*SUB A8 A9 1)) gives an inductive definition of *GPROD. Using such a definition, it is reasonable to consider such theorems as $f_1(a_1\ldots a_n) = f_1(a_1)\ldots f_1(a_n)$ for a homomorphism f_1, or $(a_1\ldots a_n)^{-1} = a_n^{-1}\ldots a_1^{-1}$. Both proofs are exactly the same in structure, step by step. Each is a natural use of induction, and each requires an additional hypothesis. For the first, this hypothesis is as follows: (EQUAL2 (F1 G H (*SUB A8 A9 N9)) (*SUB A8 (F1 G H A9) N9)). This is an explicit connection between two legal notations, according to these conventions of notation, for expressions such as $f_1(a_1)$. Thus there is actually still a need for a schema; in this case a schema to the effect that P[(*SUB A8 A9 N9)] = (*SUB A8 P[A9] N9), where P is any of a fairly large class of terms.

Leaving this incomplete discussion of the problems of notation, the question of the implementation of the method of mathematical induction will now be considered. The greatest difficulty here is an obvious one — how to decide algorithmically when to attempt a proof by induction. This decision, including the identification of the specific variable on which the induction is to be performed, is not an easy one. Surely one condition which must always be present in a theorem which is provable by this method is the appearance of a term which is defined by a definition which is inductive in form; i.e., of the form "X, Y,..., Z are special instances of a _____, and if A is a_____, the element given by the (usually simple) operation φ on A (or on A and some

other _____'s) is a_____." For instance, a common example is the positive integers, given by "1 is a positive integer, and if n is a positive integer, n + 1 is also a positive integer." *GPROD, the term discussed above, has an inductive definition.

While the presence of a term that is inductively defined is a necessity for the use of induction, it is certainly not true that use of induction is appropriate in all theorems which contain mention of such terms. Many theorems involving finite groups, for instance, are not proved by induction on the order of the group. To be short and to the point, a method has not been found to enable ADEPT to reasonably decide when to use induction.

Once such a decision is made, there is little further trouble. A supervisor along the lines of ISOLVE has been written which, when asked to prove some proposition P(n), asks ADEPT to prove P(1) and P(n) ==> P(n'). This suffices to handle most instances of induction encountered in elementary group theory. In general, more freedom is needed to specify the basic case, and a non-rigid concept of successor is needed.

Using the supervisor just described, NSOLVE, a user may decide that a proof should be done by induction, and have it done by machine. The two theorems mentioned earlier in this chapter are provable by NSOLVE, as well as the simple fact that $a_1 \epsilon\ G \Rightarrow a_1^n \epsilon\ G$ for any semigroup G. Using the same guidelines for notation described above, NSOLVE can be given problems involving, say, finite intersections of sets. For example, consider the problem that the intersection of any finite number of subgroups is itself a subgroup. This is done easily by NSOLVE if the lemma stating that the intersection of two subgroups is a subgroup is known to it. It seems reasonable to assume that this lemma would be present as a disjunct of the sufficient condition for subgroup available for use on any non-trivial problems. (The proliferation inherent in such assumptions will be discussed in the next chapter.) Given this lemma, only one extension had to be made to ADEPT. This was to have the search which sees if a line about to be put on table II is already on table I take into account (free) variables as arguments of terms on table I. That is, a

statement of the form $P(a_1)$, where a_1 is a variable, on table I carries the force of its universal generalization, and therefore should suffice to verify any statement $P(a_2)$ being put on the proof tree. Heretofore, all statements put on table I with variables happened to be either implications or equalities, and thus a simple check for equality of two statements, one on table I and one one table II, was sufficient (given that EQUAL and EQUAL2 were considered identical), since the scan would discover other possible inferences. In this problem, however, a table I statement (SUBGROUP (*SUB A A N9)) with (free) variables A and N9 has to cause verification of such lines on the proof tree as (SUBGROUP (*SUB H H 1)). Such an extension to ADEPT is clearly necessary and desirable, and can be implemented in a manner so that it will not increase running time significantly.

The preceding paragraphs have mentioned only very simple problems provable by induction. The following discussion is of a considerably harder theorem. Included here because the proof requires induction, this theorem's discussion will also serve to illustrate the problems that will arise when definitely more advanced theorems are given to mechanical proof procedures. Truly a "Pandora's box catastrophe" is very near, and only inspired heuristics will ward it off.

The theorem is that every finite group has a composition series, and first, a proof of this is outlined, in order to make the following comments more intelligible. A composition series of a group G is a series of subgroups $(A_1,\ldots, A_n)$ of G such that $A_1 = \{e_G\}$, $A_n = G$, A_{i-1} is normal in A_i and A_i/A_{i-1} is simple for $i = 2,\ldots, n$, and all the A_i's are distinct. To aid in the proof is the lemma that A/B is simple (has no non-trivial normal subgroups) if and only if B is a maximal normal subgroup of A. Proceeding to induct on the order of G, the basis step is completed by demonstrating that $(\{e_G\})$ is a composition series of length 1 for G if G has order 1. Assuming that any group of order $\leq m$ has a composition series, a group G of order $m + 1$ is shown to have one by considering its maximal normal subgroup H. H must have order $\leq m$, and therefore has a composition series $(A_1,\ldots, A_n)$, where $A_n = H$. Then

it can be shown that $(A_1, \ldots, A_n, G)$ is a composition series for G. This completes the inductive step and therefore the proof.

Now consider just a few of the details glossed over in the above outline, which cannot be omitted in a machine's proof. The definition of a maximal subgroup H of a group G includes a statement that the order of H, o(H), is strictly less than o(G). To conclude from $o(G) = m + 1$ and $o(H) < o(G)$ that $o(H) \leq m$ requires an explicit step, using one of the numerous relations connecting $<$, $\leq$, $=$, $\geq$, and $>$. Certainly these relations cannot be given to ADEPT as lemmas, but must be built (or "compiled") into the program, and in a way that will not cause proliferation of effort. Perhaps this can be done not by special-purpose matching routines for Polish notation expressions, but by a subroutine built around the use of the linear "lattice" model of the integers. Another step left out of the proof is one which might be given as a lemma for this theorem, and that is that every finite group G such that $1 < o(G)$ has a maximal normal subgroup. (How would ADEPT prove this innocent fact?) Another detail requires that the definition of order of a group state that a group of order 1 contains one element, and if the definition does not specify this element to be the identity, ADEPT must be able to prove the admittedly trivial fact that it is the identity.

In addition, demonstration of a candidate for a composition series involves a use of existential quantifiers which is not among the special cases which SOLVEX can now handle. This, of course, will be a frequent occurrence as new types of theorems are proposed for ADEPT, and some of the necessary extensions will be more difficult than others.

It is the author's hope that the preceding discussion will clearly illustrate the difficulties and considerations inherent in proofs of more complex theorems. No solutions have been delineated, partly because it is the author's contention that the harder theorems introduce radically different types of reasoning processes than those used in the types of theorems for which ADEPT was constructed.

Mostly just to prove that it could be done, even though making use of an

extension of SOLVEX written to cope with only the above theorem and making use of a "loaded" definition of composition series, a version of the preceding problem was formulated which NSOLVE successfully proved (with 26 lines and using 33 seconds of machine time). The statement of the problem is given here in mathematical notation, and the translation into Polish notation follows the spirit of the previous discussion on notation for series, iterated products, and the like.

A composition series is defined for this purpose by the inductive statement: $B = \{e_B\} \Rightarrow (B)$ is a composition series of length 1 for B; if B has a maximal normal subgroup and $(A_1, \ldots, A_n)$ is a composition series for it, then $(B_1, \ldots, B_m)$ is a composition series for B, where $m = n + 1$, $A_i = B_i$ for $i < m$, and $B_m = B$. The theorem itself is given as a conjunction of lemmas, followed by the conclusion: [G is a group and $o(G) \leq n$] $\Rightarrow$ G has a composition series. NSOLVE is told to induct on n. The lemmas used are:

i) [A is a group and $o(A) \leq 1$] $\Rightarrow A = \{e_A\}$;

ii) A is a group $\Rightarrow$ [A has a maximal normal subgroup which is itself a group, and [$o(A) \leq m + 1 \Rightarrow$ the order of the maximal subgroup of A is $\leq m$]];

iii) A has a composition series $\Rightarrow$ there exist an integer m and subgroups B_i such that $(B_1, \ldots, B_m)$ is a composition series for A.

Even these specially chosen statements caused difficulties. In particular, it was discovered that certain conjunctions on the proof tree were such that it made no sense to attempt to prove one conjunct before the other was established, and a provision was introduced to flag such a conjunct as "not discarded but not to be considered until such and such a time". This was crudely done, and is best not described in detail!

Before embarking on the next chapter for a final discussion of the future of mechanical theorem-proving as revealed in the light of the ADEPT program, one other example will be considered. Much of the reasoning used in the types of theorems which ADEPT encounters is applicable to problems previously given

to "advice-takers". In particular, an investigation was made to see what modifications would have to be made to ADEPT to solve a version of the MIKADO problem, originated by Safier[20] and discussed by Slagle.[24] It turns out that very few are necessary. The access to tables of definitions and sufficient conditions is blocked, for advice-taker problems are stated in self-contained packages. Koko, Nankipoo, Katisha, and Mikado are declared to be variables (it does not matter which kind!), and one extension is made to SOLVEX, allowing an expression of the form $(\exists a_1)\ldots(\exists a_n)P(a_1,\ldots, a_n)$, for some expression P which is a single term followed by its arguments, to be "solved" if a statement $P(b_1,\ldots, b_n)$ is on table I. The problem can then be proved by ADEPT if stated as follows:

hypotheses:

i) (Unmarried*female Katisha)

ii) (Unmarried*male Koko)

iii) (Not*think*dead Mikado Nankipoo) ==> (Can*stay*alive Koko)

iv) (Can*appear*safely Nankipoo) ==> (Can*produce Koko Nankipoo)

v) (Not*accusing Katisha Nankipoo) ==> (Can*appear*safely Nankipoo)

vi) (Not*claiming Katisha Nankipoo) ==> (Not*accusing Katisha Nankipoo)

vii) $(\exists a_1)$(Can*produce a_1 a_2) ==> (Not*think*dead Mikado a_2)

viii) (Married Katisha) ==> (Not*claiming Katisha a_1)

ix) $(\exists a_1)$(Can*marry a_1 a_2) ==> (Married a_2)

x) $(\exists a_1)$(Can*propose a_1 Katisha) ==> $(\exists a_1)$(Can*marry a_1 Katisha)

xi) (Unmarried*female a_2) & $(\exists a_1)$(Unmarried*male a_1) ==> $(\exists a_1)$ (Can*propose a_1 a_2)

conclusion:

(Can*stay*alive Koko)

In the above, a_1 and a_2 are assumed to be variables; i.e., the first six hypotheses form the first conjunction of input, and the last five hypotheses form the second.

This statement of the problem should be compared closely with that given

in Slagle.[24] The propositions involving causality used in Slagle's proof were not used with ADEPT, so it is not surprising that ADEPT's proof is shorter; in fact, it is much shorter, taking only 14 seconds instead of Slagle's 5.7 minutes. But ADEPT is not an advice-taker in a general sense, so the preceding comparisons are not too cogent. It suffices to say here that ADEPT can handle at least some advice-taker problems. In Appendix I, a more detailed discussion of advice-takers and how they differ from ADEPT will be given.

CHAPTER VIII

FUTURE POSSIBILITIES AND CONCLUSIONS

It is time to evaluate the ADEPT project and discuss possibilities for future work uncovered through insights and experience obtained from the program. To do this to best advantage requires a restatement of the purposes for which ADEPT was created. Primarily, the goal was to obtain a program which could handle a significant number of theorems of elementary group theory. Needless to say, these theorems were to be drawn from the simplest results, the foundations of the subject. These seem to fall into four main classes: i) those which require "constructions", such as the proof that a left identity is also a right identity, ii) those which follow by straightforward inferences from basic concepts, iii) those which depend to a large extent on knowledge of simple facts in number theory, such as easy results regarding finite groups, and iv) those which serve as illustrative examples. It was decided to use theorems of the second type, on the assumption that the deductive reasoning required to handle them would be most generally applicable to all kinds of theorem-proving.

As a matter of fact, consideration of the development of the simple proof that the kernel of a homomorphism is a subgroup of the domain of that homomorphism was instrumental in defining the basic structure of ADEPT. Subsequent consideration of numerous theorems of the same type led to the growth of ADEPT to its present form, complete with a number of special-purpose heuristics. Indeed, the limited goals for ADEPT as a program have been met, for nearly one hundred theorems of group theory have been proven by this computer algorithm, and the program has sufficed to provide legitimate evaluation of variations in the algorithm and their effects.

It may be objected that the proofs produced by ADEPT are sometimes inefficient, not to mention that some problems of the type that is ADEPT's specialty "stump" the program completely. In reply one has only to note that any routine which can produce proofs of all the various facts that ADEPT has established in consistently less than two minutes of computer time per proof cannot be seriously inefficient. (The isomorphism theorems if considered as three or four "proofs" each conform to this statement.) This is particularly cogent when one considers that ADEPT is written in LISP 1.5, a language not noted for execution speed. As for the similar problems with which ADEPT cannot cope, they are not numerous. Indeed, most have been discussed in earlier chapters, on the assumption that examination of shortcomings is more instructive than perusal of successes. The fact remains that the successes far outnumber the failures. More importantly, the class of theorems for which ADEPT was designed has been essentially exhausted in the course of this project. This fact has two important consequences. One is the fact that improvements made to ADEPT in order to enable it to prove those theorems of the same type which it cannot prove would not result in a program which could prove many more theorems. The other is that refinements designed to increase efficiency could only be used on the same theorems, which do not now require prohibitive amounts of effort.

More serious is the complaint that, though ADEPT can prove an impressive number of theorems, many must be stated in a "correct" manner in order for the program to be successful. In many instances this complaint is, in the author's opinion, not valid. In particular, it is maintained that a user can be reasonably asked to exercise care in deciding upon the form of statements presented to ADEPT, as least at this stage in the art of theorem-proving. For instance, choices between asterisked and non-asterisked notation, or between the various logical constants for equality are often critically important, and the user has a responsibility to aid ADEPT in this heuristic matter. The complaint is valid, however, in the case where the number, and not the form, of the hypotheses is in question. The user cannot be expected to know the minimal set of

hypotheses needed for a proof. Thus this is a definite shortcoming of the ADEPT program.

Changes designed to ameliorate this failing could be tested on the same theorems which ADEPT has already considered, by making use of the enormous number of possible ways of overspecifying the hypotheses to these problems. This is the case even if only "natural" unnecessary assumptions are introduced. But the weakness of ADEPT in coping with overspecified theorems is an indication that more serious difficulties will arise when more difficult problems, based on a broader base of known concepts, are considered. For instance, the introduction of numbers and their associated properties could complicate a proof considerably. Add to this another inefficiency which has been tolerated until now but which must be faced in the near future, namely excess branching due to the associativity axiom, and a good case has been made for conducting a full-scale investigation of more efficient theorem-proving with an eye to a much larger class of problems.

How, then, can such difficulties be overcome? Can it be by additional special-purpose heuristics, or by some general planning heuristics, or by an entirely different and more appropriate algorithm? The answer, as is so often the case, is probably a combination of the possible courses of action. Still, it seems that the most efficacious improvement would be the introduction of some kind of planning. This could take a variety of forms. In fact, some planning is already present in ADEPT. HOMOMF effectively delineates a plan to be followed to attempt to verify certain equalities (those given by FEQUAL). On a larger scale, ISOLVE represents a plan for approaching an isomorphism problem. Similar plans could be developed for other special subclasses of theorems or types of lines on the proof tree. One possibility is a special executive like ISOLVE designed for closure problems. A closure problem is one where, assuming a variable or variables are members of some set, it is to be shown that a certain function of those variables is again a member of the set. Showing sets to be subgroups or normal are problems of this kind. As was seen in Chapter V, efficiency difficulties have occurred in such theorems,

and a plan incorporated in a supervisor might be an answer to such troubles.

To resort in this manner to separate routines for each kind of problem is an abandonment of any hope of developing a general-purpose problem solver, except for a program to perform the very trivial deductions which form the core of most proofs. Such despair may be justified, but this cannot be argued conclusively at this date. For one thing, there are more general ways to attempt planning. One such possibility would not be hard to try with ADEPT, and probably should be tried soon. This particular planning heuristic will now be elaborated upon in some detail.

In Chapter III the phrase "explore consequences of all terms..." was used, to indicate a call to SCANW to match against lines on the proof tree all instances of definitions of the terms in question as well as all other hypotheses and accumulated "knowledge" on table I at the time of the call. In the early stages of a problem this amounted to a planning heuristic, for due to the selective placement of entries on table I, very little was on table I except entries associated with the terms in question. Thus the "plan" was to apply to a line on the proof tree only facts known about terms, etc. of that one particular line. As the proof progressed, and more entries appeared on table I, this heuristic gradually "disappeared", giving way to a procedure of a much more "brute-force" character. Why not employ this heuristic at all times, thus incidentally eliminating the need for selective placement of entries on table I? This could be done by "indexing" all table I entries, according to their origin. Obviously there would be the original hypotheses, and the lemmas which were added during the proof, but more importantly, those entries which were definitions of terms could be "tagged" by that term, and singled out from table I whenever that term was spotted in a line of the proof tree. Thus for every line of status PNT, when it was scanned table I could be divided into two parts — a select part including all definitions of terms or constants in the line being scanned (and perhaps the original hypotheses), and the remainder of table I. Clearly the select subset would be used alone by the scan at first, and if no success or progress resulted, possibly but not

necessarily the remainder would be considered.

In conjunction with the changes now being suggested, two details should be implemented. One concerns a more involved check of the constants of a line being examined. At present, for each constant of the line, a search is made through table I for entries which define it. To be precise, a constant A is defined by an entry of the form (TERM A ...), and the resulting action is the placement on (the select portion of) table I of the proper instance of the definition of TERM. This procedure handles set and function constants adequately, but very seldom provides any information regarding constants which are members of sets. To remedy this, the procedure could be supplemented by a search of table I for terms defining not only the constants of the line of status PNT, but also all set constants A from table I entries of the form (MEMBER A1 A), where A1 is a constant of the line of the proof tree being examined. (For asterisked notation this procedure would put on (the select portion of) table I an instance of the definition of *TERM when A1 was found in an entry of the form (MEMBER A1 (*TERM ...)).) This would provide a more complete determination of which entries of table I are relevant to a given proof tree line of status PNT.

In addition, it is suggested as a result of experience, that certain entries of the select portion of table I be given even higher priority. This might be too difficult to implement to be practical, but the idea is this. Consider the case where a line $\underline{n}$ of the proof tree has given rise to a reduction $\underline{n}_1$ due to an application of detachment from an implication of table I in logical class $\underline{m}$. Suppose further that other table I entries in the same logical class could have caused substitutions in line $\underline{n}$, if this were not prevented by the restriction described in connection with "progress heuristic" B. The suggestion is that if the same "inhibited" substitutions are now possible in a line of the proof tree which is either $\underline{n}_1$ itself or a reduction of $\underline{n}_1$, that these should be done first, and given a definite priority. As an illustration of this, the reader is urged to refer back to Problem V-C, and to note that implementation of this suggestion would remove the inefficiency

associated with that proof.

The heuristics just described, by analyzing more completely the content of a line of the proof tree, including, in the last suggestion, an element of analysis of how the line originated, clearly would implement a greater degree of "awareness" to the program, so that it could be maintained more cogently that ADEPT would be performing as though it knew where it was in a proof. On a more prosaic level, proofs would run faster, if only because just a subset of table I would be used by SCANW except in unusual cases. Some proofs would no doubt be shorter from the point of view of the number of lines on the proof tree, but it is less clear that increased efficiency would manifest itself noticeably in that manner. Another important advantage would be gained — namely that the separation of a select portion of table I before scanning would enable ADEPT to cope successfully with problems involving a longer total table I; i.e., problems with more hypotheses or problems involving a broader base of concepts. There is no reason to believe that single lines of the proof tree in such problems would have more terms and constants than at present; thus the size of the select portion of table I for any given line should remain small and manageable.

While the previous heuristics would enable ADEPT to handle more assumptions, hypotheses, etc., they do nothing to alleviate a similar problem — more branches on the proof tree. As theorems become more complex, and ADEPT acquires a broader base of "knowledge", lines of the proof tree will have greater numbers of reductions. One example of this was alluded to briefly in the last chapter — sufficient conditions will have to be disjunctions with many clauses. Under these conditions, it will be impossible to expect ADEPT to consider reductions of lines in the order of their generation, as is presently done. Some choice will have to be made, on the basis of the particular context of the theorem at a point in its proof. One isolated suggestion was made in the preceding discussion concerning a proposed priority for substitutions "delayed" by prior applications of detachment. A crude choice could be based on length or complexity of a reduction, with "simpler" reductions being

processed first. Such a heuristic might work in some cases, but it is the author's opinion that any such completely syntactic heuristic would have limited effectiveness. Perhaps it would be best to use a complexity heuristic only in cases where one alternative is dramatically simpler than the others, as sometimes happens when a "correct" substitution results in a "collapsing" of a lengthy expression to a very simple one.

Other possible criteria for ordering reductions of a line are based on the history of the reductions. For instance, since generation of reductions by detachment has been seen to be correlated with progress, perhaps reductions generated in this manner should be considered first. Another possibility which may have a desirable effect is to keep track of the logical class of the table I entry which is used in generating each reduction of a line on the proof tree. Then a list of all logical classes of table I entries used in the steps from the head of the tree to any particular branch would be available. (When an implication is split and the reduction is an instance of the consequent of the parent line, there is no associated logical class, but neither is there any other reduction necessitating a decision.) Given this history of a line, priority could be given to reductions generated due to axioms, etc., in a new logical class, or in a little-used logical class, or perhaps in a logical class that had not been used recently. This would be a manifestation of the heuristic stating that all the hypotheses may be needed for a successful proof.[16]

Another suggestion is to make use of a grouping of terms into classes according to their generality. For instance, subgroup is probably the most general term which defines a set in group theory. In contrast, an inverse image can be considered to be a fairly specific set. With such a grouping, which would not have to be too refined, preference could be given to reductions generated using table I entries obtained from definition instances of the more specific terms. The rationale behind this proposal is shown in many of the examples of the earlier chapters. Many theorems involve subgroups, for instance, but the presence of the instances of the definition of subgroup

must not cause a lot of extra branches, as it presently does in ADEPT. So this is a heuristic which might help the program to cope with overspecified problems and problems with many hypotheses.

Some of the above suggestions may be futile hopes. One important fact is that because of this project, it is or may soon be possible to test such heuristics. Using ADEPT or a slightly (not greatly) improved ADEPT and the more difficult of the theorems already proved by ADEPT along with some of a slightly harder nature, it should be possible to obtain significant evaluation of the effectiveness of such schemes, and of different combinations and priorities in combinations of such schemes.

Incidentally, the worth of some of these proposed heuristics will vary according to the degree to which a hypothesis is liable to be necessary. For simple problems, this means that techniques used to cope with theorems stated with deliberate "red-herring" hypotheses will vary from those used for more ideally stated problems. For harder problems, it means that because of the larger number of hypotheses per problem, and therefore the lessened likelihood that any one hypothesis will be "correct" for a given step, and because of hypotheses included due to uncertainty over their relevance, different criteria for evaluation may be needed than were optimum for simpler problems. To give an example, the suggested "approval" for use of axioms of varied logical class is obviously a heuristic which could lead to wasted effort in the presence of irrelevant hypotheses.

A couple of other details involving ordering of reductions are of more certain usefulness. The MODEL heuristic can be applied to lines of the main proof tree, not to accept or completely reject lines as it does for those of status ST, but to give an indication of the likelihood of proving a reduction of the form $a_1 \in A$. And it hardly needs to be mentioned that when one reduction happens to be a conjunction, which will itself be of status NILL, that this branch should be ordered according to the worst evaluation of the two co-conjuncts. Such pessimism is only common sense since both conjuncts would have to be established in order to verify their predecessor.

It should be noted that adoption of schemes for ordering reductions raises questions about the order in which the total proof proceeds. Some heuristics would appear to decrease the need for complete scans, but if SCANW is not called often, a new procedure will be necessary to adequately process lines of status ST, the subordinate trees. It will be necessary to specify when to end processing of a selected reduction. If processing is ended in the same way as that of a line of status PNT is terminated at present, where does the program go next? The answer nearest to the spirit of the present version is illustrated as follows: Suppose line (4), say, has just been processed, and the next line of status REL able to take on status PNT is line (6). Perhaps line (6) is a reduction of line (3), and another reduction of (3), say line (9), is of higher priority. Then simply interchange lines (6) and (9) and proceed as usual.

Some other heuristics which can properly be called planning heuristics are suggested by features of previous work in problem-solving by computer which were not included in ADEPT. For instance, the difference-operator techniques of Newell, Shaw, and Simon[15] have an obvious application for the establishment of identities. For example, when an equality requires a switching of constants, as in a proof that a set is abelian, the axiom concerning the inverse of a product may be useful, as in Problem V-E. While it would seem impractical to develop a complete GPS-like routine to handle such steps, certainly some sort of comparison between halves of an equality could be developed and used to some degree in the choice of the next step to be made by ADEPT, as well as a means of evaluating progress. A more specific heuristic is suggested by a particular example in one of the GPS papers.[15] There, planning is done by "abstracting" the problem; since the domain of the example is the predicate calculus, this could be done by ignoring connectives and the order of the variables. ADEPT has need of such an approach in its treatment of associativity. Here the abstracting would be accomplished by ignoring grouping; i.e., "remove the parentheses". There are strong reasons for formally considering composition as a strictly binary operation, for this

simplifies the task of formalizing and programming group-theoretic algorithms tremendously. But it has become obvious that at certain times in some problems, ADEPT must be able to "stand back" and consider the lines of the proof tree in this more abstract form, thus making optimum use of the known hypothesis of associativity. Surely it would not be difficult to decide upon a precise form for this special-purpose planning heuristic and then to implement it within ADEPT.

What are the implications of the implementation of planning heuristics for the methods of determining "progress" in ADEPT? Is heuristic B, which is presently in use, suitable for a program altered in ways such as have been considered above? The answer is that heuristic B is not sophisticated enough, but more constructive comments are not as easy to make. Of course, if a priority scheme is adopted to order reductions of lines, the method of assigning priorities will probably also be of some application for a determination of progress. For instance, if terms are classed according to specificity, generation of a reduction of a line by detachment using a table I line arising from the definition of a very general term may well not be considered grounds for declaring "progress" or for putting a line of status REL into status REL1. Conceivably, with good enough planning and ordering heuristics, there would be no need for status REL1. Or if determination of order of reductions depended strongly upon which were generated by detachment, perhaps a progress heuristic should not be based so strongly on application of detachment. It does seem, however, in the light of results obtained so far with ADEPT, that uses of detachment must be given some kind of special consideration in either planning or evaluating steps in any extension of ADEPT.

Of course, if special-purpose executives, like ISOLVE, are constructed, as was suggested above for closure problems, the need for lower-level planning and evaluation of progress will decrease markedly. In fact, since a large number of theorems considered thus far with ADEPT are closure problems, it might be a worthwhile short-term project to develop such an executive in order to evaluate the special-purpose approach. In contrast, development of planning

heuristics and more sophisticated progress heuristics should be done in light of more theorems than those considered so far in the course of this project.

It should also be noted that other existing heuristics besides those connected with "progress" may be rendered unproductive by the adoption of planning techniques. Just as an example, the heuristic of substituting in only one half of an equality on the proof tree may be found to be of no further value. As in all this discussion, precise statements about effects can not be made until a particular combination and implementation of some or all of the possible new heuristics is adopted and fixed.

As mentioned at various times in this report, part of the effort of extending ADEPT will have to be devoted to development of specific subroutines to augment the existing program. These are not heuristics, but merely capabilities for different types of problems. SOLVEX will have to be able to handle expressions of the form $(\exists a_1)Pa_1$ for a broader class of expressions Pa_1, including expressions themselves headed by an existential quantifier. The need for a routine to discover "constructions" or expansions has been seen in a number of examples. Such a routine will not be easy to develop. It will have to include heuristics, quite possibly ones very reminiscent of Newell, Shaw, and Simon's GPS.[15] This routine will govern those cases where equalities expressed using EQUAL (including those handled by the simplification routines associated with PUTON2) should be applied in the unnatural direction. It should also be capable of either performing or setting up intermediate steps so that ADEPT can perform substitutions such as $a_1a_2^{-1} \leftarrow (a_2a_1^{-1})^{-1}$, which were seen to figure in some of the theorems covered in Chapter VI.

As difficult as it will be to devise a "construction" subroutine, an even more difficult task will be to decide algorithmically when to use it. Here the realm of the unexplored has been reached. No longer does the orientation of ADEPT seem so productive, but still it is infinitely preferable to the orientation of the complete predicate calculus procedures. Here open, creative minds are needed to cross a gap of ignorance. Similar situations have come up before in this report. When should induction be used? What

procedure can isolate the variable on which the induction will take place? For proofs by contradiction, efficiency virtually demands prior selection of the hypothesis most likely to be contradicted by the consequences of assuming the denial of the conclusion. Even more difficult is the question of when to abandon proof of a proposed theorem and try to construct a counter-example for it. (Incidentally, a good "give-up" heuristic could conceivably make it practical to try all of the methods of proof on a given problem, thus eliminating the need to choose between proof by induction, contradiction, etc., without increasing the total amount of effort involved by a very high factor.) And ultimately, could a computer program generate likely candidates for theorems?

Returning from this dash into the world of the relatively distant future of theorem-proving by computer, consider problems of only slightly greater difficulty than those already done by ADEPT. In particular, consider briefly how the concepts of orbit of an element and order of an element are naturally used in the proof of the Sylow theorems, which need only mention order of the group in their statements. This is not an isolated example; to prove more advanced theorems one must know what method to use, and the method often involves introduction of other concepts. In part, one learns from experience, but if only from experience the result is a group theorist like the author, who is unable to prove a theorem unless he has seen the proof of a similar one! But how does the mind of the creative mathematician work? Could a scholar outline an algorithm suitable for a computer? Is it not more likely that a programmer attacking this problem will meet the same frustration as Dr. A. L. Samuel met when he found that expert checkers players did not understand how they played the game?

Thus the author firmly believes that radically different approaches are used in the thought processes of human mathematicians as they work on harder theorems, and therefore different algorithms will be needed to prove harder theorems by computer. Indeed, though it could be broadened to prove more theorems, ADEPT is already to the point where its orientation has approached the

limit of its productiveness, and fresh ideas are needed in order to produce efficient proofs of harder theorems. There will, however, be need for ADEPT, or rather for an improved ADEPT, in future problem-solving programs no matter what their organization, for harder theorems will have sub-theorems which are precisely the type of problem for which ADEPT was constructed. In addition, part of the reasoning process of harder proofs, particularly once a general method of proof has been chosen, will involve just those very logical inferences that ADEPT can handle.

In summary, what has been done is this. Starting with a desire to capture in an algorithm suitable for a machine the human thought-processing used in elementary group theory, a program was developed. Before this development could be accomplished, a decision had to be made between two possible orientations — the "mechanical" and the "heuristic". The former had been considered more extensively in previous projects. On the surface it seemed that a combinatorial, manipulative approach with a simple but powerful theory behind it was ideally suited to a computer program, as well as providing the satisfying theoretical "completeness" which is a consequence of the theory. But the previous efforts had shown that what seemed ideal in theory was not at all ideal in practice, and in fact discouraging barriers were encountered in the realm of efficiency. At the same time, the heuristic approach had not been shown to be necessarily superior, but at least it was a viable alternative. Its main advantage is that the programmer can avail himself of human experience in a direct way. By introspection and investigation of mathematical reasoning as seen in the literature, an attempt can be made to understand how the human mathematician is able to reduce what is combinatorially a gigantic problem to manageable size. (It must not be overlooked that even this "size" is not small when one takes into account the "experience of the ages", the store of knowledge and effort of centuries before.) Thus the investigation takes on a two-fold interest — not only is there the attempt to construct a working algorithm, but there is the endeavor of understanding in part the techniques of the brain, and formalizing those steps of logical deduction that are done

literally without thinking.

There is no claim here that what is best for the mind is best for the machine. It is certainly conceivable that the optimum way for a computer to prove theorems bears little relation to human thought. But at the present time it is not known what this might be. One "non-human" orientation has already been seen to not be the answer. Meanwhile, progress can be made, both in developing algorithms and in understanding the problem itself, by consideration of what people know about what is, after all, a uniquely human ability at this time.

Pointed in this direction, the development of ADEPT began. First and foremost, as has been seen, the program has been successful, though this fact may have been obscured at times by the discussion of ADEPT's shortcomings and what could be done next. Not only have non-trivial theorems been proved by a computer program, but a huge number of insights into the natures of specific problems and types of problems as they are approached algorithmically have been obtained. Furthermore, heuristics have been discovered and evaluated, now that a suitable vehicle for such experimentation is in existence. It will be instructive to list briefly, in a single compact list, the heuristics used in the course of this project.

1) Implementation, through the use of specially introduced logical constants for equality (EQUAL and FEQUAL), of restrictions on substitutions and of different methods of processing lines.
2) Substitutions allowed in one side only of an equality (EQUAL or EQUAL2) on the proof tree.
3) Performance of related substitutions simultaneously, implemented through the use of a "logical class" for table I entries.
4) Processing of lines headed by an existential quantifier by special cases, as opposed to a uniform procedure.
5) Selective placement of definition instances on table I,

done as terms are encountered in the course of a proof.

6) Restricted matching allowed for expressions consisting of just one free variable.

7) "Compiled" application of common identities as a line is put on the proof tree, sometimes resulting in simplification of the line, and other times resulting in additional substitution instances of the line.

8) The MODEL heuristic, allowing rejection of some new subordinate trees started by an expression of the form (MEMBER A1 A), using a "lattice" model of the sets involved.

9) Various progress heuristics, in particular, heuristics A and B, involving consideration of detachment.

Some of these heuristics have proved more successful than others. For instance, numbers 1), 3), 6), 7), and 8) seem to have proved their value conclusively, and can be retained in essentially their present form. Others, as implied in the preceding discussion of possible improvements, may be found to be of less importance or worth. And obviously, new heuristics are still needed.

Though ADEPT never was intended to embody a "complete" procedure, a moment's reflection will show that some of the above heuristics further limit ADEPT's theoretical power. For instance, sometimes "progress" is declared erroneously, as in the overspecified version of Problem V-A. It would, of course, be a trivial matter for ADEPT to be programmed so that once it became "stuck", it lifted these heuristic restrictions and redid the problem in a more comprehensive manner. (It is a less trivial matter to "know" when the program is "stuck"!) But such a recovery procedure is of little interest, as is obvious from the original abandonment of a "complete" procedure. A decrease in ADEPT's theoretical power caused by a heuristic is of no concern, if the heuristic permits more efficient proofs for more theorems in general, where "more", of course, has to refer to a great many more theorems than are "foiled" by the proposed heuristic. Certainly the heuristics included in

ADEPT conform to this policy. They have enabled ADEPT to perform acceptably, and what is more, this report has been candid about those examples which point out shortcomings of the heuristics.

The question can be raised: Why stop here? Many possibilities, both immediate and more long-range, exist for continued work. The answer is not simply that one has to stop someplace; this is in reality a natural break-point. This is not to say that what exists of ADEPT is perfected; it is not even to say that the discussion of future possibilities suggests answers to all of the problems raised in the earlier chapters. However, as stated earlier, no simple alterations or extensions to the program will enable it to handle a significant number of new theorems. Quite the contrary; unless major additions are created, only better proofs of theorems already provable will result.

Therefore, this project (and this report) comes to an end. Those difficulties encountered and overcome in the course of these efforts can now be handled routinely in the future, freeing further researchers in the area of heuristic problem-solving so that they may be able to concentrate more clearly on the harder problems ahead.

One final comment should be made. It has been suggested to the author by Tim Hart that heuristics developed by experiments such as have been carried out with ADEPT may be able to be restated in a form compatible with "complete" procedures, and incorporated into them, perhaps to the end that a basically combinatorial approach will surpass the basically heuristic, empirical approach. This is in no way an argument against the ADEPT project. If such a reliance on heuristics which can only (or at least more easily) be discovered through the more natural orientation to theorem-proving used in ADEPT, can then be transferred to a more mechanical program, the dividing line between the two approaches will virtually disappear, and advantage will have been taken of the best virtues of both. To the author, though, the most important contributions will still be the heuristics. Hopefully, ADEPT will pave the way for important discoveries of that nature.

APPENDIX I

OTHER WORK IN THEOREM-PROVING

This discussion of related work in theorem-proving has been placed after the body of the report on ADEPT for two reasons. The first is that before understanding the approach used in ADEPT it would not be clear which previous projects were relevant as background. Secondly, this history can now be supplied with comparisons and comments on the author's contributions.

The appearance of theorem-proving as a branch of artificial intelligence was inevitable. As soon as researchers began to explore the potentialities of digital computers for tasks other than numerical computation, and more specifically, as soon as non-numerical tasks requiring "intelligence" were seen to be legitimate subject areas for computer programs, artificial intelligence was in existence. It is not the intention of this commentary to define the proper bounds of this area of computer science, or to become involved in a harangue over the use of the word "intelligence". The fact is that it became possible to conceive of a digital computer, properly programmed, making decisions and performing tasks that in human experience necessitated "thought" and reasoning of a higher level than merely mechanical operations. Such possibilities were highly exciting. If achieved, computers could become extremely powerful and valuable aids, much more so than they are because of their computational capabilities alone. As a by-product, scientists would have a useful model of a class of human mental processes. Consequently, artificial intelligence projects were undertaken. Not surprisingly, the first tasks explored in these endeavors contained a large element of mechanical procedure and only a small amount of required "thinking". Excellent examples were found in games and in the theorems of the early chapters of Russell and Whitehead's

monumental Principia Mathematica. Closely related, though not usually put under the category of artificial intelligence, was the effort to make rough translations from one language to another.

Work in all these areas became more sophisticated. An admirable achievement was made by Dr. A. L. Samuel in his master-level checkers program,[21] a milestone in artificial intelligence, and an excellent example of a special-purpose program designed to do only one task, and to do it well. Programs for computerized formal mathematics also became more ambitious, and the early programs for doing mathematical logic were joined by such successes as Gelernter's geometry program [5,6] and Slagle's symbolic integration program. [23] Because of its self-contained nature, at least in the early stages of the subject, and its ease of formalization, the theory of abstract groups became a frequent subject of theorem-proving programs. But the successes were followed by a long series of slight improvements, without any real break-through.

Meanwhile, a number of investigators delineated a problem area of "logical deductive reasoning", independent of subject matter. Often with ambitious claims of generality and applicability, programs were constructed to handle a wide variety of questions. Some, as GPS — Newell, Shaw, and Simon's General Problem Solver,[15] were actually crude models of human rational thought. By necessity, these general programs contained heuristics — procedures designed to cope with certain cases or combinations of events in a manageable way; procedures believed to be of wide applicability but not necessarily guaranteed to be always productive. Heuristic, general routines needed to be tested on some particular syllabus. Often this syllabus was drawn from the theorems of some branch of mathematics.

In any event, the attempt to prove theorems of formal mathematics by computer is not done as an end in itself, but as a step toward the achievement of artificial intelligence, in the hope of achieving a much greater usefulness of digital computers. At the present day such a hope is far from realization, and so work in artificial intelligence continues to concern itself with

theorems, games, etc.

Henceforth this discussion will concentrate only on the area of theorem-proving by computer. Progress is being made constantly, and the state of the art is no doubt rather more advanced than can be seen from the literature, due to the time lag involved in publishing. In particular, projects are known to be under way at MIT and at Carnegie Tech at this time, and undoubtedly other work is in progress elsewhere. However, only what has been published will be presented here.

The fundamental difference between the mechanical and heuristic approaches to theorem-proving has already been maintained. A familiarity with the general organization of a "complete" proof procedure has been assumed, and ADEPT itself is an example of the heuristic, albeit special-purpose, approach. What has not been done is to point out how far other projects have gone, and to compare ideas used in ADEPT with some of those used in earlier programs.

Beginning with the mechanical, combinatorial approach, what can be said about the capabilities of the many programs of this nature which have been written, and what means have been employed to reduce combinatorial explosion? The most encouraging report is that of Wos, Robinson, and Carson,[29] which relates that a program (hereafter called the "Wos program") solved Problem V-E (a group each of whose elements is of order 2 is abelian) in as little as 5.18 seconds. The main heuristic used was the so-called "set of support strategy", which can be described precisely only in terms of a formal logical system, but which amounts to giving special consideration to the conclusion and special hypotheses (in this case the hypothesis that $a_1 a_1 = e$ for all a_1 in the group). Since the proof procedure works forward, attempting to generate a contradiction from the hypotheses and the denial of the conclusion, this means that the program looked for contradictory sets of statements only among sets of statements containing clauses derived from the two lines which were singled out. In ADEPT, such a heuristic would essentially say that any proof must make use of all special hypotheses, where the user, as in the case of the

Wos program, tells the program which hypotheses are to be so treated.

In the Wos paper, the interplay between the set of support strategy and an arbitrary bound imposed on the complexity of statements generated is discussed. The latter involves a guess as to the complexity of the proof of a proposed theorem. Not surprisingly, in the presence of an accurate guess the set of support strategy is of little help. But its worth is clear in the absence of such a guess, for with the strategy a 37.5 second proof is obtained when no bound is specified, while a 411 second proof with a conservative bound and no proof at all with no bound is obtained without the set of support strategy. ADEPT's performance on this problem has already been discussed. Its "working backward" elminates much of the need for the set of support strategy, inasmuch as most irrelevant lemmas in ADEPT are automatically excluded. It was seen that ADEPT could not solve the problem without being explicitly told about a consequence of the hypothesis, namely $a_1 = a_1^{-1}$ for all a_1 in the group. As described in Chapter V, ADEPT proved two versions of the restated problem, and one took 22 seconds while the other took 82 seconds. Only 18 lines on the proof tree were needed in the latter case, as opposed to over a thousand for Wos' worst case. In conclusion it should be pointed out that this problem is quite a bit more suitable for a combinatorial theorem-prover than many which have been solved by ADEPT.

The Wos program, of course, profited by the experience of its predecessors. It uses a variation of the Herbrand procedure that was formalized by J. A. Robinson,[19] using a single rule of inference, resolution, which is rather well-suited for the task of mechanization. Robinson, in turn (as is excellently recounted in his 1963 paper[18]), drew upon improvements outlined by Davis and Putnam[3] and others. All in all, much effort has been spent on developing an extremely efficient formalization of a complete proof procedure, greatly refining Herbrand's original scheme. For instance, properties common to the syntax of contradictory sets have been utilized. But refinements in the procedure are not heuristics, and are not even subject-oriented.

Hao Wang has done a great deal of investigation[26,27,28] along similar

lines. For instance, he has improved procedures by developing separate algorithms for statements of different syntactic forms; i.e., different patterns of quantifiers in prenex normal form. Wang's papers contain the most persuasive arguments for the combinatorial as opposed to the heuristic approach. Unfortunately, his arguments for the preferability of a sound theory and complete procedure seem based on a faith that such an approach can, with effort, be made successful. No program has been demonstrated to date to support this faith. If it is a justifiable hope, Wang's arguments would be hard to ignore; if, as this author believes, the hope is forlorn, then the theoretical advantages of a complete procedure are irrelevant.

This concludes an admittedly brief account of combinatorial theorem-proving programs. Before a discussion of specifically heuristic problem-solvers is begun, mention should be made of the work of a man who had no interest in computer programs for formal mathematics, but who nonetheless has completely captured the spirit and worth of heuristics in his pedagogical approach. Prof. G. Polya's books on problem-solving[16,17] are "musts" for anyone who is interested in the procedures used by people while doing "formal" reasoning.

In the field of artificial intelligence, perhaps no researchers are better known than Newell, Shaw, and Simon. Their programs — first the Logical Theorist (LT)[14] and then the General Problem Solver (GPS)[15] — have had an enormous impact upon this area of computer science. (Perhaps, as Newell himself notes,[13] this impact has been too great, causing a lack of innovation in approaching artificial intelligence.) LT was conceived and implemented a decade ago, yet its ideas have reappeared in most subsequent heuristic problem-solvers. Partly, this is due to Newell, Shaw, and Simon's emphasis on simulation of human behavior, because, being concerned with uncovering models of human thought processes, these researchers specified algorithms and heuristics which seem very natural, for they attempt to capture our (the human's) way of approaching problem-solving. For a number of reasons, work in theorem-proving often turns to draw upon human experience, and in so doing,

emulates LT. For instance, an introduction to the gross inefficiency inherent in a blind "working forward" approach as found in the early combinatorial theorem-provers leads one to the observation that people often work backward from the conclusion, either completely so as in LT, almost so as in ADEPT, or at least keeping the goal in mind to guide steps taken, as can be said of GPS. "Working backward" is for many problem areas a high-level heuristic, and a very effective one for reducing irrelevant effort. It was one of the main points made by the LT project.

Adoption of a "working backward" type of procedure usually involves the use of a matching subroutine, and ADEPT has one just as did LT. Now, as then, this subroutine accounts for a significant fraction of the effort expended by the entire program. Newell, Shaw, and Simon experimented with heuristics (the similarity tests) designed to facilitate matching, but had limited success. As they pointed out, those possible matches which heuristics can easily recognize as not worth sending to the matching subroutine are just those which the subroutine itself rejects after a minimal amount of work. In ADEPT, very little effort was spent trying to improve the matching process. The only feature incorporated for this purpose was the special-purpose matching routine to handle the checking of a fixed list of common axioms and identities by PUTON2. Here, use of a quick check for key terms common to axioms and the proof tree line was sufficient to expedite the matching of these particular statements.

Particularly on a superficial level, there is a great deal of similarity between LT and ADEPT. Both use ad hoc methods chosen to handle a particular problem area. Since the "task environment" of LT, propositional calculus, is semantically simpler (i.e., of less content) than group theory, it is not strange that LT is a less complicated program. Both ADEPT and LT employ non-complete procedures. LT seems more formally structured than ADEPT, but this is due largely to the nature of the problem area. Still, ADEPT does not use "methods", tried in turn on lines of the proof tree. Instead, each axiom is allowed to operate on the proof tree line, in turn (or at a special time, in the case of the common axioms and identities which have been singled out),

in whatever way its syntax permits. Modus ponens plays a prominent role in each program, though the specific use of this rule of inference takes different forms in each. However, on a more detailed level, again partly due to the nature of the problem areas, ADEPT is more resourceful than LT. Not only does it contain more heuristics, but it can be extended to include or test other special-purpose heuristics which might be developed. It makes more use of the context of the problem, as, for instance, in the examination of terms and constants of a proof tree line, and in the recognition of table I lines defining constants used in the proof, thus enabling instances of definitions to be used to better advantage.

The creators of LT discovered a property of heuristics that also has been observed in ADEPT, namely that they enable a program to reach a "plateau" of performance, beyond which it is very difficult to progress without very different heuristics. This was mentioned in Chapter VIII as primarily a property of the problem area, the author maintaining that harder problems necessitate new methods. This differentiation of problems is harder to observe in systems oriented around the predicate calculus, and perhaps this was one reason which led Newell, Shaw, and Simon to leave the impression that what is needed is a more powerful set of general-purpose heuristics as opposed to different heuristics for different classes of problems.

In any event, new heuristics were (and are) needed, and GPS followed LT. GPS is a much more ambitious project, with many psychological implications. Intended to be a model of problem-solving in many areas, GPS was tested on problems in propositional calculus and trigonometry. Its basic organization is much too well known to need recounting here; suffice to say it employs certain "goal types" to apply operators that reduce differences between known statements and desired statements. The operators and differences must be specified and characterized for each problem area to which GPS is directed; the methods or "goal types" are fixed, and conform roughly to means-end analysis. Clearly ADEPT has no similarity to GPS in general organization. Yet GPS has been mentioned in passing in this report, primarily in Chapter VIII,

because it appears that attempts to remedy some of ADEPT's weak points may be able to profit from insights gained in the GPS work. Comparing present with desired results has an obvious analogue in the establishment of identities, and a proposed low-level planning heuristic for use with GPS, namely abstraction of the problem, seems applicable to more efficient treatment of associativity, as described previously in Chapter VIII.

While discussing the work of Newell, Shaw, and Simon, a paper of Newell and Ernst[13] should be mentioned as giving a provocative overview of heuristic programming. Its perspective on program organization appears that it might be very useful to anyone contemplating new routines for problem-solving. While by no means as far-reaching as Minsky's Steps Toward Artificial Intelligence,[11] the paper is quite general, and helps to unite the field, as have Minsky's papers.

Perhaps the project nearest in spirit to ADEPT is the Geometry-Theorem Proving Machine of Gelernter.[5,6] First reported in 1958, the geometry machine is a heuristic theorem-prover designed specifically for one branch of mathematics. In fact, like ADEPT, it handles a subclass of the elementary theorems of its subject, namely theorems involving congruences, parallel lines, and equality or inequality of segments and angles. Over 50 such theorems were successfully proved by the geometry machine, and it became a useful vehicle for testing special-purpose heuristics. Thus it was the first program to successfully handle a significant number of theorems from a particular branch of formal mathematics. Like ADEPT, it used a "working backward" procedure which was ad hoc in formalization, and not complete. Even some of the particular heuristics used in the two programs are closely related. Both programs single out certain deductions to be made immediately on a proof tree line. Also, both programs make use of a diagram.

At this point of similarity, the divergence between the two projects becomes most clear. Gelernter's geometry machine is totally dependent upon its diagramming feature. This is in no way a fault of the program; it is to its credit that this semantic model proves so powerful for its problem domain.

However, anyone acquainted with group theory will hardly be surprised that the semantic model used in ADEPT plays a relatively minor role. Of course, the relative importance of one heuristic is hardly a major factor for evaluation, but its consequences are, for the geometry machine, due to the success of the diagram heuristic, did not have to develop any other means for controlling proliferation of effort, and indeed was incapable of handling even the simplest of theorems without recourse to a diagram. With ADEPT, the basic organization had to be more clever to allow efficient proofs, and, as argued throughout this report, whatever success has been obtained in this regard reflects progress toward a theorem-prover which "understands" where it is in a proof, and is able to make use of context. The formal part of the geometry machine has no such capabilities, for they were not needed. Thus both the geometry machine and ADEPT have been successful programs for a branch of formal mathematics, but ADEPT was forced to contain more sophisticated techniques, and the research involved in their development is more likely to be of value in the future.

The idea of a diagram, or more generally, a semantic model, is a very fruitful one, and was first propounded by Minsky.[10] A use for a model was found in group theory, as seen in ADEPT's MODEL heuristic, and there is every reason to believe that other subjects will lend themselves to other realizations of this general idea. Certainly future workers in theorem-proving must be alert to the possibility of favorable uses of such a feature.

The other projects relevant to this report are the general question-answering systems that come under the heading of "advice-takers". The advice-taker was conceived by McCarthy,[7] and it is oriented to the development of a system which would be capable of learning and working with any subject area, including "everyday situations". To do this, it was designed so that it was very general, handling all facts in the same way, and what is more, in a rather simple form which was intended to facilitate use of the system by many people. Because of this generality, the advice-taker is a victim of ambition. Like the combinatorial theorem-provers, it is ill-suited

to the introduction of techniques based on context, and heuristics intended to become an integral part of the program's structure. Like heuristic programs, it lacks formal theory, forcing a reliance on ad hoc programming which in this case is thwarted by the proposed generality of the system. Admittedly, McCarthy did not see it in this light; rather the advice-taker was to profit by an ability to accept heuristics in exactly the same way as the data base and questions. But the method of introduction of statements makes use of a variation on formal logic which retains some characteristics of English, just as in ADEPT. This means that heuristics must also be stated in this form, which appears to be a harsh requirement. In theory, a huge bootstrapping operation could be undertaken, but this appears beyond the capacity of present systems.

In order to deal in detail with "everyday situations", McCarthy proposed that formal predicates of ability and cause be introduced,[9] along with statements expressing properties of and relations between these predicates (such as a statement saying that if A can cause an event a, and if event a causes event b, then A can cause event b). Sample problems proposed for the advice-taker were stated in terms of these predicates. In some cases, such as the Mikado problem[20,24] (considered at the end of Chapter VII), the use of these predicates was not necessary to express the essential content of the problem, as was seen in the version of the Mikado problem posed for ADEPT. Still, the expanded version serves as an illustration of a possible way of working with a problem-solver of great generality. One should note, however, that the character of the problem is not altered by the presence of additional predicates and axioms; the result is simply a more complex problem of the same type.

The actual algorithm which is the advice-taker is very simple, and it is small wonder that ADEPT has enough deductive power to "answer" versions of some advice-taker problems. However, it would be erroneous to claim that ADEPT can do anything that an advice-taker can. In fact, two versions of an advice-taker have been created, one by Slagle[24] and one by Black,[1] and

both have interesting capabilities not useful for formal mathematics and therefore not included (and difficult to insert) in ADEPT. Slagle's DEDUCOM is imbedded in the LISP interpreter, and this simple fact allows great flexibility in use of symbol processing and arithmetic capabilities of LISP. Introduction of even simple arithmetic to ADEPT would be rather awkward. More importantly, both advice-takers allow multiple answers to questions, and the solutions of problems created by this feature are non-trivial.

In terms of ADEPT, the ability to provide multiple answers means free variables on table II, something which is not provided for at present. Black gives an excellent account of the difficulties these variables create and their implications for program organization. The matching routines become more complex and, as is to be expected, a larger amount of effort per problem is needed in order to find all answers. There is no doubt that multiple answer capability is necessary to a general deductive system; it is also true that it is an ability not needed in formal mathematics such as group theory.

Early versions of Black's program fell into infinite loops at times because they did not check whether a line being put on the proof tree was already on the tree. ADEPT checks this in PUTON2, and it does not seem to be a time-consuming process, as Black feared. Other loops occurred from axioms with hidden pitfalls, like (IMPLIES (NOT (NOT A1)) A1), which would allow to a line A2 on the proof tree the reduction (NOT (NOT A2)), which in turn would have a reduction (NOT (NOT (NOT (NOT A2)))), etc. ADEPT encountered a very similar difficulty in Problem V-F due to the hypothesis $f_1(a_1) = f_1(a_2) \Longrightarrow a_1 = a_2$. With ADEPT, the problem was circumvented by use of FEQUAL. Black also found the form and order of hypotheses to be critical in some problems, as did Slagle. To make deductive algorithms independent of such factors would almost necessarily introduce an unbearable degree of inefficiency.

Black experimented with some heuristics to improve his advice-taker. One was a bound on the proof tree by means of various length restrictions. This purely syntactic strategy was less than satisfying. He discussed possible schemes for stopping or avoiding loops. He also proposed grouping axioms and

hypotheses according to their main terms or connectives. This is done in the present implementation of ADEPT. For one thing, in order to facilitate PUTON2's application of common identities, special lists are kept of all homomorphisms and factor groups, so that a separate search of table I for this information is unnecessary. Table I itself is split into two parts, as might be deduced from Figure 4. All the equalities and implications are on one part, and only this part is used directly by SCANW. However, such streamlining is hardly of crucial importance to the efficiency and success of the program. (Warren Teitelman[25] has also explored some modifications of Black's advice-taker.)

This concludes the account of various projects related to the ADEPT project and how ADEPT contrasts with them. No mention has been made of other important areas in which many feel that significant progress must be made in order for any large advances to be made in artificial intelligence. One is learning and another is organization of data and concepts. ADEPT is not a learning program, and would not be a suitable foundation for one. Learning, of course, refers to machine assimilation of knowledge, as opposed to increasing the "knowledge" of a program by the addition of more routines by a programmer. Slagle points out that his DEDUCOM[24] (and therefore ADEPT) "learns" when given more information, such as a new axiom or definition. In a limited sense this is true, particularly from the advice-taker orientation. Yet it would appear that real machine learning must involve programmed induction and generalization or adaptation facilities.

As for data organization, even less is known on how to proceed. List structures have been standard up until now. Indeed, all of the programs mentioned in this chapter under the general heading of "heuristic", including ADEPT, have been written using list processing languages. Without these languages these programs would have been almost impossible to write. Still, one wonders if a fresh view toward data organization could be helpful.

APPENDIX II

THEOREMS PROVED BY ADEPT

```
1)  (GROUP G)
     NIL
    (SEMIGROUP G)

2)  (AND (KERNEL K (F1 G H)) (HOMOMORPHISM (F1 G H)))
     NIL
    (SUBGROUP K G)

3)  (AND (KERNEL K (F1 G H)) (HOMOMORPHISM (F1 G H)))
     NIL
    (NORMAL K G G)

4)  (AND (CENTER C G) (ASSOC G))
     NIL
    (SUBMONOID C G)

5)  (CENTER C G)
     NIL
    (ABELIAN C G)

6)  (AND (CENTER C G) (AND (GROUP G) (SUBSET G C)))
     NIL
    (ABELIAN G G)

7)  (AND (CENTER C G) (AND (SUBSET H C) (GROUP G)))
     NIL
    (NORMAL H G G)

8)  (AND (SUBSET H G) (AND (ABELIAN G G) (GROUP G)))
     NIL
    (NORMAL H G G)

9)  (AND (SUBSET H G) (AND (ABELIAN G G) (GROUP G)))
     NIL
    (ABELIAN H G)

10) (AND (INTERSECTION I H J) (AND (SUBGROUP H G) (SUBGROUP J G)))
     NIL
    (SUBGROUP I G)

11) (AND (INTERSECTION I H J) (AND (NORMAL H G G) (NORMAL J G G)))
     NIL
    (NORMAL I G G)

12) (AND INTERSECTION I H J) (AND (SUBGROUP J G) (NORMAL H G G)))
     NIL
    (NORMAL I J G)
```

```
13)  (AND (IMAGE I (F1 G H)) (AND (HOMOMORPHISM (F1 G H)) (GROUP G)))
      NIL
     (SUBGROUP I H)

14)  (AND (CONJUGATE D A1 H G) (AND (GROUP G) (SUBGROUP H G)))
      NIL
     (SUBGROUP D G)

15)  (AND (AND (SUBGROUP H G) (LCOSET D A1 H G)) (MEMBER A1 H))
      NIL
     (SUBSET D H)

16)  (AND (SUBSET A G) (AND (LCOSET D (*IDENTITY G) A G) (GROUP G)))
      NIL
     (SUBSET D G)

17)  (AND (SUBSET A G) (AND (LCOSET D (*IDENTITY G) A G) (GROUP G)))
      NIL
     (SUBSET D A)

18)  (AND (SUBSET A G) (LCOSET D (*IDENTITY G) A G))
      NIL
     (SUBSET A D)

19)  (AND (GROUP G) (AND (SUBSET A G) (AND (LCOSET D (*IDENTITY G) A G)
        (RCOSET E (*IDENTITY G) A G))))
      NIL
     (SUBSET D E)

20)  (AND (GROUP G) (LCOSET D A1 G G))
      NIL
     (AND (SUBSET G D) (SUBSET D G))

21)  (AND (GROUP G) (AND (LCOSET D A1 G G) (RCOSET E A1 G G)))
      NIL
     (SUBSET D E)

22)  (AND (SUBGROUP H G) (LCOSET D A1 H G))
      NIL
     (MEMBER A1 D)

23)  (AND (SUBGROUP H G) (NORMALIZER D H G))
      NIL
     (SUBSET H D)

24)  (NORMALIZER J H G)
      NIL
     (NORMAL H J G)

25)  (GROUP G)
      NIL
     (NORMAL G G G)

26)  (UNITSET E (*IDENTITY G))
      NIL
     (NORMAL E G G)

27)  (UNITSET E (*IDENTITY G))
      NIL
     (SUBGROUP E G)
```

```
28)  (AND (COMMUTATOR A1 A2 A3 G) (ASSOC G))
     (EQUAL2 (*PROD A2 A3 G) (*PROD A3 A2 G))
     (EQUAL A1 (*IDENTITY G))

29)  (AND (COMMUTATOR A1 A2 A3 G) (AND (ASSOC G) (AND (ABELIAN G G)
        (AND (MEMBER A2 G) (MEMBER A3 G)))))
     NIL
     (EQUAL A1 (*IDENTITY G))

30)  (AND (AND (COMMUTATOR A1 A2 A3 G) (COMMUTATOR A4 A3 A2 G)) (ASSOC G))
     NIL
     (EQUAL (*INVERSE A1 G) A4)

31)  (AND (AND (COMMUTATOR A1 A2 A3 G) (COMMUTATOR A4 A3 A2 G)) (ASSOC G))
     NIL
     (INVERSE A4 A1 G)

32)  (AND (GROUP G) (AND (MEMBER A1 G) (MEMBER A2 G)))
     NIL
     (AND (EXISTS A3 (EQUAL (*PROD A1 A3 G) A2)) (EXISTS A3 (EQUAL
        (*PROD A3 A1 G) A2)))

33)  (SUBGROUP H G)
     NIL
     (IMPLIES (AND (MEMBER A1 H) (MEMBER A2 H)) (MEMBER (*PROD A1
        (*INVERSE A2 G) G) H))

34)  (GROUP G)
     (EQUAL2 A1 (*INVERSE A1 G))
     (ABELIAN G G)

35)  (GROUP G)
     (AND (EQUAL (*PROD A1 A1 G) (*IDENTITY G)) (IMPLIES (EQUAL
        (*PROD A1 A1 G) (*IDENTITY G)) (EQUAL2 A1 (*INVERSE A1 G))))
     (ABELIAN G G)

36)  (AND (HOMOMORPHISM (F1 G H)) (KERNEL K (F1 G H)))
     (IMPLIES (MEMBER A1 K) (EQUAL A1 (*IDENTITY G)))
     (ONETOONE (F1 G H))

37)  (AND (AND (HOMOMORPHISM (F1 G H)) (KERNEL K (F1 G H))) (ONETOONE
        (F1 G H)))
     NIL
     (IMPLIES (MEMBER A1 K) (EQUAL A1 (*IDENTITY G)))

38)  (AND (AND (AND (LCOSET D A1 H G) (SETINV I D G)) (SUBGROUP H G))
        (GROUP G))
     NIL
     (RCOSET I (*INVERSE A1 G) H G)

39)  (AND (AND (LCOSET D A1 H G) (LCOSET E A2 H G)) (AND
        (SUBGROUP H G) (ASSOC G)))
     (AND (MEMBER (*PROD (*INVERSE A2 G) A1 G) H) (EQUAL
        (*PROD (*INVERSE A1 G) A2 G) (*INVERSE (*PROD (*INVERSE A2 G)
        A1 G) G)))
     (AND (SUBSET D E) (SUBSET E D))

40)  (AND (AND (HOMOMORPHISM (F1 G H)) (AND (MEMBER A1 G) (INVPNT I
        (F1 G H A1) (F1 G H)))) (KERNEL K (F1 G H)))
     NIL
     (LCOSET I A1 K G)
```

```
41)  (AND (NORMAL H G G) (SUBSET K G))
      NIL
     (NORMAL H K G)

42)  (AND (AND (SUBSET A B) (SUBSET B G)) (AND (CENTRAL C A G)
        (CENTRAL D B G)))
      NIL
     (SUBSET D C)

43)  (AND (HOMOMORPHISM (F1 G H)) (AND (RIMAGE J K (F1 G H))
        (INVIMAGE I J (F1 G H))))
      NIL
     (SUBSET K I)

44)  (AND (AND (EPIMORPHISM (F1 G H)) (CENTER C G)) (AND
        (RIMAGE I C (F1 G H)) (CENTER D H)))
      NIL
     (SUBSET I D)

45)  (AND (EPIMORPHISM (F1 G H)) (IMAGE I (F1 G H)))
      NIL
     (SUBSET H I)

46)  (AND (HOMOMORPHISM (F1 G H)) (AND (ABELIAN G G) (IMAGE I
        (F1 G H))))
      NIL
     (ABELIAN I H)

47)  (AND (AND (HOMOMORPHISM (F1 G H)) (SUBGROUP K G)) (RIMAGE I K
        (F1 G H)))
      NIL
     (SUBGROUP I H)

48)  (AND (NORMAL K G G) (AND (RIMAGE I K (F1 G H)) (EPIMORPHISM
        (F1 G H))))
      NIL
     (NORMAL I H H)

49)  (AND (AND (HOMOMORPHISM (F1 G H)) (NORMAL K G G)) (AND (RIMAGE I K
        (F1 G H)) (IMAGE J (F1 G H))))
      NIL
     (NORMAL I J H)

50)  (AND (HOMOMORPHISM (F1 G H)) (AND (SUBGROUP K H) (INVIMAGE D K
        (F1 G H))))
      NIL
     (SUBGROUP D G)

51)  (AND (HOMOMORPHISM (F1 G H)) (AND (NORMAL K H H) (INVIMAGE D K
        (F1 G H))))
      NIL
     (NORMAL D G G)

52)  (AND (SUBGROUP H G) (AND (FACTORGROUP B K G) (SUBFGRP C K H G)))
      NIL
     (SUBSET C B)

53)  (AND (SUBGROUP H G) (AND (FACTORGROUP B K G) (SUBFGRP C K H G)))
      NIL
     (SUBGROUP C B)
```

```
54)  (AND (NORMAL H G G) (AND (FACTORGROUP B K G) (SUBFGRP C K H G)))
      NIL
     (NORMAL C B B)

55)  (GROUP G)
     (IMPLIES (MEMBER A1 G) (EQUAL (F1 G G A1) (*IDENTITY G)))
     (HOMOMF (F1 G G))

56)  (GROUP G)
      NIL
     (ISOMORPHIC G G)

57)  (ISOMORPHISM (F1 G H))
      NIL
     (ISOMORPHIC G H)

58)  (ISOMORPHISM (F1 G H))
      NIL
     (ISOMORPHIC H G)

59)  (AND (GROUP G) (AND (ISOMORPHISM (F1 G H)) (ISOMORPHISM (F2 H K))))
      NIL
     (ISOMORPHIC G K)

60)  (AND (ISOMORPHISM (F1 G H)) (ISOMORPHISM (F2 H K)))
      NIL
     (ISOMORPHIC K G)

61)  (AND (FACTORGROUP D E G) (AND (UNITSET E (*IDENTITY G)) (GROUP G)))
      NIL
     (ISOMORPHIC G D)

62)  (AND (FACTORGROUP D E G) (UNITSET E (*IDENTITY G)))
      NIL
     (ISOMORPHIC D G)

63)  (AND (AND (FACTORGROUP D K G) (KERNEL K (F1 G H))) (EPIMORPHISM
        (F1 G H)))
      NIL
     (ISOMORPHIC D H)

64)  (AND (AND (FACTORGROUP D K G) (KERNEL K (F1 G H))) (EPIMORPHISM
        (F1 G H)))
      NIL
     (ISOMORPHIC H D)

65)  (AND (AND (FACTORGROUP D K H) (FACTORGROUP E I H)) (AND
        (INTERSECTION I H K) (SUBGROUP K H)))
      NIL
     (ISOMORPHIC D E)

66)  (AND (AND (FACTORGROUP D K H) (FACTORGROUP E I H)) (AND
        (INTERSECTION I H K) (SUBGROUP K H)))
      NIL
     (ISOMORPHIC E D)

67)  (AND (AND (FACTORGROUP D I G) (FACTORGROUP E K H)) (AND
        (EPIMORPHISM (F1 G H)) (INVIMAGE I K (F1 G H))))
      NIL
     (ISOMORPHIC D E)
```

```
68)  (AND (AND (FACTORGROUP D I G) (FACTORGROUP E K H)) (AND
       (EPIMORPHISM (F1 G H)) (INVIMAGE I K (F1 G H))))
     NIL
     (ISOMORPHIC E D)

69)  (AND (AND (FACTORGROUP A H G) (FACTORGROUP B K G)) (AND
       (SUBFGRP C K H G) (FACTORGROUP D C B)))
     NIL
     (ISOMORPHIC A D)

70)  (AND (AND (FACTORGROUP A H G) (FACTORGROUP B K G)) (AND
       (SUBFGRP C K H G) (FACTORGROUP D C B)))
     NIL
     (ISOMORPHIC D A)

71)  (AND (ABELIAN G G) (GROUP G))
     NIL
     (ISOMORPHIC G G)
     (IMPLIES (MEMBER A1 G) (EQUAL (F1 G G A1) (*INVERSE A1 G)))
              (overriding GENFCN)

72)  (AND (MEMBER A1 G) (GROUP G))
     NIL
     (ISOMORPHIC G G)
     (IMPLIES (MEMBER A2 G) (EQUAL (F1 G G A2) (*PROD (*PROD A1 A2 G)
       (*INVERSE A1 G) G)))
              (overriding GENFCN)

73)  (AND (GROUP G) (FACTORGROUP D H G))
     NIL
     (ISOMORPHIC G D)
              (fails on one-to-one, thus proving epimorphic)

74)  (AND (FACTORGROUP D K G) (AND (FACTORGROUP E I H) (AND
       (RIMAGE I K (F1 G H)) (EPIMORPHISM (F1 G H)))))
     NIL
     (ISOMORPHIC D E)
              (fails on one-to-one, thus proving epimorphic)

75)  (AND (LCOSET D A1 H G) (AND (LCOSET E A2 H G) (AND (ASSOC G) (AND
       (MEMBER A1 D) (SUBSET D E)))))
     NIL
     (MEMBER (*PROD (*INVERSE A2 G) A1 G) H)

76)  (AND (SETINV I H G) (AND (NORMAL H G G) (SUBGROUP H G)))
     NIL
     (AND (SUBSET H I) (SUBSET I H))

77)  (AND (SETINV I H G) (SUBGROUP H G))
     NIL
     (SUBGROUP I G)

78)  (AND (SETINV I H G) (AND (NORMAL H G G) (AND (SUBGROUP H G) (ASSOC G))))
     NIL
     (NORMAL I G G)

79)  (AND (CENTER C G) (ABELIAN G G))
     NIL
     (SUBSET G C)
```

```
80)  (HOMOMORPHISM (F1 G H))
     (AND (AND (EQUAL (*SEQ (A9 N2) 1 A) (A9 1)) (EQUAL (*SEQ (F9 A B
        (A9 N2)) 1 B) (F9 A B (A9 1)))) (AND (EQUAL (*SEQ (A9 N2)
        (*SUCCESSOR N3) A) (*PROD (*SEQ (A9 N2) N3 A) (A9 (*SUCCESSOR N3))
        A)) (EQUAL (*SEQ (F9 A B (A9 N2)) (*SUCCESSOR N3) B) (*PROD (*SEQ
        (F9 A B (A9 N2)) N3 B) (F9 A B (A9 (*SUCCESSOR N3))) B))))
     (FEQUAL (F1 G H (*SEQ (A1 N1) N G)) (*SEQ (F1 G H (A1 N1)) N H))

81)  (HOMOMORPHISM (F1 G H))
     (AND (AND (EQUAL (*SEQ A8 A9 1 A) (A8 A9 1)) (EQUAL (*SEQ A8 A9
        (*SUCCESSOR N1) A) (*PROD (*SEQ A8 A9 N1 A) (A8 A9 (*SUCCESSOR N1))
        A))) (EQUAL (F9 A B (A8 A9 N1)) (A8 (F9 A B A9) N1)))
     (FEQUAL (F1 G H (*SEQ A1 A1 N G)) (*SEQ A1 (F1 G H A1) N H))

82)  (AND (SEMIGROUP G) (MEMBER A1 G))
     (AND (EQUAL (*EXP A9 1 G) A9) (EQUAL (*EXP A9 (*SUCCESSOR N1) G)
        (*PROD (*EXP A9 N1 G) A9 G)))
     (MEMBER (*EXP A1 N G) G)

83)  (AND ($SUBGROUP (H 1) G) ($SUBGROUP (H (*SUCCESSOR N)) G))
     (IMPLIES (AND ($SUBGROUP A C) ($SUBGROUP B C)) ($SUBGROUP
        (*INT A B) C))
     ($SUBGROUP (*FININT H N) G)

84)  (AND ($NORMAL (H 1) G G) ($NORMAL (H (*SUCCESSOR N)) G G))
     (IMPLIES (AND ($NORMAL A C C) ($NORMAL B C C)) ($NORMAL (*INT A B) C C))
     ($NORMAL (*FININT H N) G G)

85)  (NULL)
     (AND (AND (IMPLIES (AND (GROUP A) (LEP (ORDER A) 1)) (EQUAL A
        (*UNITSET (*IDENTITY A)))) (IMPLIES (GROUP A) (AND
        ($HAS*MAX*NORM*SUB A) (AND (GROUP (*MAX*NORM*SUB A))
        (IMPLIES (LEP (ORDER A) (*SUCCESSOR N1)) (LEP (ORDER
        (*MAX*NORM*SUB A)) N1)))))) (IMPLIES ($HAS*COMP*SERIES A)
        (EXISTS N3 (EXISTS (B N4) (COMP*SERIES ((B N4) N3) A)))))
     (IMPLIES (AND (GROUP G) (LEP (ORDER G) N)) ($HAS*COMP*SERIES G))

86)  (AND (AND (UNMARRIED*FEMALE N7) (UNMARRIED*MALE N5)) (AND (IMPLIES
        (NOT*THINK*DEAD N8 N6) (CAN*STAY*ALIVE N5)) (AND (IMPLIES
        (CAN*APPEAR*SAFELY N6) (CAN*PRODUCE N5 N6)) (AND (IMPLIES
        (NOT*ACCUSING N7 N6) (CAN*APPEAR*SAFELY N6)) (IMPLIES
        (NOT*CLAIMING N7 N6) (NOT*ACCUSING N7 N6))))))
     (AND (IMPLIES (EXISTS N1 (CAN*PRODUCE N1 N2)) (NOT*THINK*DEAD N8 N2))
        (AND (IMPLIES (MARRIED N7) (NOT*CLAIMING N7 N1)) (AND (IMPLIES
        (EXISTS N1 (CAN*MARRY N1 N2)) (MARRIED N2)) (AND (IMPLIES (EXISTS N1
        (CAN*PROPOSE N1 N7)) (EXISTS N1 (CAN*MARRY N1 N7))) (IMPLIES (AND
        (UNMARRIED*FEMALE N2) (EXISTS N1 (UNMARRIED*MALE N1))) (EXISTS N1
        (CAN*PROPOSE N1 N2)))))))
     (CAN*STAY*ALIVE N5)
              (N5 = Koko
               N6 = Nankipoo
               N7 = Katisha
               N8 = Mikado)
```

APPENDIX III

LISTING OF THE PROGRAM

The reader who attempts to use the listings included in this appendix will find that they do not conform in all details with the descriptions given in the body of this report. Any discrepancies of this sort were introduced solely to make the report easier to comprehend, and do not really misrepresent the actual operation of the program.

The various LISP functions are made into a system by means of a file such as COMPIL. Following execution of this file, the user may call ADEPT as a LISP function of two arguments. The first of these is to be either "NIL" or "MODEL", and determines whether or not the MODEL heuristic is to be in effect. The second argument is to be either "NIL" or "INDUCT", and is the present means whereby NSOLVE can be called, when use of mathematical induction is desired. (Warning: Even with the special LISP system available to the author and extensive use of REMOB,[8] it was not possible to load all the functions into the system at once by use of COMPIL as presented here. This would presumably be the case with most LISP systems.)

The file DEFNS DATA contains the sufficient conditions and definitions used by the author. The two remaining lists in this file represent certain of the built-in axioms and lattice information for use with the MODEL heuristic, respectively.

INDEX TO LISTING

ADDANTEC, 162
ADDCONSQ, 162
ADEPT, 141
ALLMATCH1, 169
ALLSUBSTS, 165
ALLVBLES, 164
ANDARGS, 162
ANDEND, 169
ANDTERMS, 169
APPENDM, 162
ARGOF1, 166
ASSOCM, 165
ASTERISKED, 165
ATMS, 168
CADDAAR, 166
CADDDAAR, 166
CADDDAR, 166
CHECKMODEL, 160
CHNGW, 165
COMPIL, 172
COMPOSITION, 155
CONNECTIVE, 164
DEFNS DATA, 170-171
DELPR, 168
DELTF, 168
FIFTH, 165
FREEVBLE, 168
GENFCN, 154
GENSET, 160
GENSUBST, 164
GENVBLE, 168
GFIND, 169
GMATCH, 163
GMATCH1, 163
GMEMB, 169
GTFRM, 161
HFM, 159
HOMOMF, 153
IMPMATCH, 163
INVM, 158
INVM2, 158
INVREDO, 167
INVSUB, 167
ISOLVE, 142
MASK, 162
MATCH1, 162
MEET, 160
MEMOF1, 167
MIRGEA, 168
MODEL MAKE, 160
MONOMF, 156
NEXT, 166
NOLOOK, 165
NOWWHICH, 160
NSOLVE, 143
ON, 164
ONLY, 153
ONLYMEMB, 168
ON2, 149
ON2AVER, 162
PMATCH, 165
PRODARGS, 167
PUTN2, 149
PUTON1, 148
PUTON2, 149
PUT2, 149-150
RESRV, 164
ROFV, 165
ROFW, 166
RPLCE, 166

RPLCEF, 168
RPLC1, 169
RPTSUB, 167
SCANW, 146-147
SCNX, 152
SIFT, 158
SIXTH, 165
SLVX, 153
SOLVE, 144-145
SOLVEX, 151
SOLVX, 143
SOLVXP, 153
SPREAD, 165
SUPSUB, 164
SUPXEC, 164
SWMEMB, 160
TACK, 168
TERMS, 164
VBLES, 164
VERIFY, 157
WLDFN, 155
WLDFN1, 155

```
                          ADEPT        LISP
DEFINE(((ADEPT(LAMBDA(QM5 WH)                                                  00010
(PROG(W1 W X Y Z1 Z2 Z3 Z4 N Z6 Q QF1 U U1 QL1 QL2 XX CX CX1 LON                00020
  QM1 QM2 QM3 QM4 QM6 FQM2 FQM3 FQM4 U2 U3 ZN VV1 VV2 CL Z4A Z4B)              00030
  (SETQ N (QUOTE NILL))                                                        00040
  (FILESEEK(QUOTE DEFNS)(QUOTE DATA))                                          00050
  (SETQ X (READ))                                                              00060
  (SETQ Y (READ))                                                              00070
  (SETQ XX (READ))                                                             00080
  (SETQ QM1 (READ))                                                            00090
  (FILEENDRD(QUOTE DEFNS)(QUOTE DATA))                                         00100
H1  (SETQ Z1 (QUOTE((NILL.A)(NILL.B)(NILL.C)(NILL.D)(NILL.E)                   00110
  (NILL.G)(NILL.H)(NILL.I)(NILL.J)(NILL.K)(NILL.M))))                          00120
  (SETQ Z2 (QUOTE((NILL.F1)(NILL.F2)(NILL.F3)(NILL.F4)(NILL.F5)                00130
  (NILL.F6)(NILL.F7)(NILL.F8)(NILL.F9))))                                      00140
  (SETQ Z3 (QUOTE((NILL.A1)(NILL.A2)(NILL.A3)(NILL.A4)(NILL.A5)                00150
  (NILL.A6)(NILL.A7)(NILL.A8)(NILL.A9)(NILL.A10)(NILL.A11)(NILL.A12))))        00160
  (SETQ ZN (QUOTE((NILL.N)(NILL.N1)(NILL.N2)(NILL.N3)                          00170
  (NILL.N4)(NILL.N5)(NILL.N6)(NILL.N7)(NILL.N8)(NILL.N9))))                    00180
  (SETQ Q NIL)                                                                 00190
(SETQ QM2 NIL)(SETQ QM3 NIL)(SETQ QM4 NIL)(SETQ QM6 NIL)                       00200
  (SETQ QL1 NIL)                                                               00210
  (SETQ QL2 NIL)                                                               00220
  (SETQ Z4 1)(SETQ Z4A 1)(SETQ Z4B 1)                                          00230
  (SETQ Z6 NIL)                                                                00240
  (SETQ LON NIL)                                                               00250
  (SETQ VV1 NIL)(SETQ VV2 NIL)                                                 00260
  (SETQ W1 NIL)                                                                00270
  (SETQ W NIL)                                                                 00280
  (SETQ U2 (RDFLX()))                                                          00290
  (SETQ U1 (ALLVBLES(VBLES U2)))                                               00300
  (SETQ Z1 (RESRV U1 Z1))                                                      00310
  (SETQ Z2 (RESRV U1 Z2))                                                      00320
  (SETQ Z3 (RESRV U1 Z3))                                                      00330
  (SETQ ZN (RESRV U1 ZN))                                                      00340
  (SETQ CL (ALLVBLES(TERMS U2 (ALLVBLES(VBLES U2)))))                          00350
  (SETQ U3 (RDFLX()))                                                          00360
  (SETQ U (RDFLX()))                                                           00370
  (COND((EQUAL(CAR U)(QUOTE IMPLIES))(GO H1A)))                                00380
  (SETQ U1 (ALLVBLES(VBLES U)))                                                00390
  (SETQ Z1 (RESRV U1 Z1))                                                      00400
  (SETQ Z2 (RESRV U1 Z2))                                                      00410
  (SETQ Z3 (RESRV U1 Z3))                                                      00420
  (SETQ ZN (RESRV U1 ZN))                                                      00430
H1A  (COND((EQUAL(CAR U2)(QUOTE AND))(SETQ U1 (PUTON1(CONS Z4A N) U2)))      00440
    (T  (SETQ U1 (PUTON1(CONS Z4A Z4B) U2))))                                  00450
  (SETQ Z4B (ADD1 Z4B))                                                        00460
  (COND((NULL U3)(GO H1B)))                                                    00470
  (SETQ U2 U3)                                                                 00480
  (SETQ U3 NIL)                                                                00490
  (GO H1A)                                                                     00500
H1B  (COND((MEMBER(CAR U)(QUOTE(ISOMORPHIC ONEONE)))                           00510
      (SETQ U1 (ISOLVE U)))                                                    00520
    ((NULL WH)(SETQ U1 (SOLVE U)))                                             00530
    (T (SETQ U1 (NSOLVE U))))                                                  00540
  (GO H1)                                                                      00550
)))))   STOP)))                                                                00560
```

```
                          ISOLVE        LISP
DEFINE(((ISOLVE(LAMBDA(U)(PROG(U1 EQN FV FV2)                                  00010
X1  (COND((NULL CL)(GO X2)))                                                   00020
  (SETQ U1 (GTFRM(CAR CL)(QUOTE Y) NIL Z4B))                                   00030
  (SETQ CL (CDR CL))                                                           00040
  (SETQ Z4B (ADD1 Z4B))                                                        00050
  (GO X1)                                                                      00060
X2  (SETQ FV (CONS VV1 VV2))                                                   00070
  (SETQ FQM2 QM2)(SETQ FQM3 QM3)(SETQ FQM4 QM4)                                00080
  (SETQ EQN (RDFLX()))                                                         00090
  (COND((EQUAL(CAR L)(QUOTE ONEONE))(GO B)))                                   00100
  (SETQ U1 (MEMOF1 VV1 (QUOTE ISOMORPHISM)))                                   00110
X4  (COND((NULL U1)(GO X3))                                                    00120
    ((EQUAL(CDAAR U1)(CDR U))(GO F6)))                                         00130
  (SETQ U1 (CDR U1))                                                           00140
  (GO X4)                                                                      00150
X3  (COND((NULL EQN)(SETQ EQN (GENFCN(CDR U)))))                               00160
  (COND((NULL EQN)(GO F1)))                                                    00170
  (PRINT EQN)                                                                  00180
  (SETQ VV1 (CAR FV))(SETQ VV2 (CDR FV))                                       00190
  (SETQ QM2 FQM2)(SETQ QM3 FQM3)(SETQ QM4 FQM4)(SETQ QM6 NIL)                  00200
  (SETQ U1 (PUTON1(CONS Z4A Z4B) EQN))                                         00210
  (SETQ Z4B (ADD1 Z4B))                                                        00220
  (SETQ FV2 (CONS VV1 VV2))                                                    00230
  (COND((EQUAL(CAR EQN)(QUOTE IMPLIES))(GO A)))                                00240
  (SETQ U1 (WLDFN(CDR EQN)))                                                   00250
  (COND((NULL U1)(GO F2)))                                                     00260
  (SETQ U1 (CAADR EQN))                                                        00270
  (SETQ VV1 (CAR FV2))(SETQ VV2 (CDR FV2))                                     00280
  (SETQ QM2 FQM2)(SETQ QM3 FQM3)(SETQ QM4 FQM4)(SETQ QM6 NIL)                  00290
  (GO A1)                                                                      00300
A   (SETQ U1 (CAAR(CDADDR EQN)))                                               00310
A1  (SETQ U1 (SOLVE(LIST(QUOTE HOMOMF)(CONS U1 (CDR U)))))                     00320
  (COND((NULL U1)(GO F3)))                                                     00330
  (SETQ Z6 NIL)(SETQ LON NIL)                                                  00340
  (SETQ W1 NIL)                                                                00350
  (SETQ W NIL)                                                                 00360
  (SETQ VV1 (CAR FV2))(SETQ VV2 (CDR FV2))                                     00370
  (SETQ QM2 FQM2)(SETQ QM3 FQM3)(SETQ QM4 FQM4)(SETQ QM6 NIL)                  00380
  (COND((EQUAL(CAR EQN)(QUOTE IMPLIES))(GO P)))                                00390
P1 (SETQ U1 (SOLVE(LIST(QUOTE ONTO)(CONS(CAADR EQN)(CDR U)))))                 00400
  (COND((NULL U1)(GO F4)))                                                     00410
  (SETQ Z6 NIL)(SETQ LON NIL)                                                  00420
  (SETQ W1 NIL)                                                                00430
  (SETQ W NIL)                                                                 00440
  (SETQ VV1 (CAR FV))(SETQ VV2 (CDR FV))                                       00450
  (SETQ QM2 FQM2)(SETQ QM3 FQM3)(SETQ QM4 FQM4)(SETQ QM6 NIL)                  00460
B   (SETQ U1 (MONOMF EQN N))                                                   00470
  (COND((NULL U1)(GO F5)))                                                     00480
C   (PRINT(QUOTE QED))                                                         00490
  (RETURN T)                                                                   00500
F1  (PRINT(QUOTE(CANNOT FIND RELATING FUNCTION)))                              00510
  (GO H1)                                                                      00520
F2  (PRINT(QUOTE(CANNOT PROVE FUNCTION WELL DEFINED)))                         00530
  (GO H1)                                                                      00540
F3  (PRINT(QUOTE(CANNOT PROVE FUNCTION HOMOMORPHIC)))                          00550
  (GO H1)                                                                      00560
F4  (PRINT(QUOTE(CANNOT PROVE FUNCTION EPIMORPHIC)))                           00570
  (GO H1)                                                                      00580
F5  (PRINT(QUOTE(CANNOT PROVE FUNCTION MONOMORPHIC)))                          00590
H1  (RETURN F)                                                                 00600
F6  (PRINT U)                                                                  00610
  (PRINT(QUOTE BECAUSE))                                                       00620
  (PRINT(LIST(QUOTE ISOMORPHISM)(CAAR U1)))                                    00630
  (GO C)                                                                       00640
P   (SETQ U1 (CADDR EQN))                                                      00650
  (SETQ VV2 (SUBST U1 EQN VV2))                                                00660
  (SETQ EQN U1)                                                                00670
  (GO P1)                                                                      00680
)))))    STOP))))))                                                          00690
```

```
                        NSOLVE        LISP
DEFINE(((NSOLVE(LAMBDA(U)(PROG(U1 FV)                                          00010
  (SETQ FV (CONS VV1 VV2))(SETQ FQM2 QM2)(SETQ FQM3 QM3)(SETQ FQM4 QM4)        00020
  (SETQ U1 (SUBST 1 (QUOTE N) U))                                              00030
  (PRINT(QUOTE(ATTEMPTING PROOF BY INDUCTION)))                                00040
  (SETQ U1 (SOLVE U1))                                                         00050
  (COND((NULL U1)(GO F1)))                                                     00060
  (PRINT(QUOTE(BASIS STEP PROVED)))                                            00070
  (SETQ LON NIL)                                                               00080
  (SETQ Z6 NIL)(SETQ W1 NIL)(SETQ W NIL)(SETQ VV1 (CAR FV))(SETQ VV2           00090
  (CDR FV))(SETQ QM2 FQM2)(SETQ QM3 FQM3)(SETQ QM4 FQM4)(SETQ QM6 NIL)         00100
  (SETQ U1 (PUTON1(CONS Z4A Z4B)(SUBST(QUOTE EQUAL)(QUOTE FEQUAL) U)))         00110
  (SETQ Z4B (ADD1 Z4B))                                                        00120
  (SETQ U (SUBST(QUOTE(*SUCCESSOR N))(QUOTE N) U))                             00130
  (SETQ ZN (RESRV(QUOTE(N)) ZN))                                               00140
  (SETQ U1 (SOLVE (CONS(QUOTE INDUCTION) U)))                                  00150
  (COND((NULL U1)(GO F2)))                                                     00160
  (PRINT(QUOTE(INDUCTIVE STEP PROVED)))                                        00170
  (PRINT(QUOTE QED))                                                           00180
  (RETURN T)                                                                   00190
F1  (PRINT(QUOTE(CANNOT COMPLETE BASIS STEP)))                                 00200
  (RETURN F)                                                                   00210
F2  (PRINT(QUOTE(CANNOT COMPLETE INDUCTIVE STEP)))                             00220
  (RETURN F)                                                                   00230
)))))   STOP)))))                                                              00240
```

```
                        SOLVX         LISP
DEFINE(((SOLVX(LAMBDA(U)(PROG(U1 U2 U3 S)                                      00010
  (COND((ON(CADDR U) ZN)(GO B)))                                               00020
  (GO EXTEND)                                                                  00030
B  (SETQ U1 (CADDDR U))                                                        00040
B1A  (COND((EQUAL(CAR U1)(QUOTE EXISTS))(GO B1))                               00050
    ((CONNECTIVE(CAR U1))(GO B2)))                                             00060
  (SETQ U2 (MEMOF1 VV1 (CAR U1)))                                              00070
B4A  (COND((NULL U2)(GO B3)))                                                  00080
  (SETQ U3 (GMATCH(CDR U1)(CAR U2)))                                           00090
  (COND((NULL U3)(GO B4)))                                                     00100
  (SETQ U3 (PUTN2 NIL (CONS(CAR U1)                                            00110
    (CAR U2))(CADDDAR U)))                                                     00120
  (COND((NULL U3)(GO B4)))                                                     00130
  (RETURN U3)                                                                  00140
B4  (SETQ U2 (CDR U2))                                                         00150
  (GO B4A)                                                                     00160
B1  (SETQ U1 (CADDR U1))                                                       00170
  (GO B1A)                                                                     00180
B3  (SETQ U2 (MEMOF1 VV2 (QUOTE IMPLIES)))                                     00190
B3A  (COND((NULL U2)(GO B2))                                                   00200
    ((EQUAL(CAADAR U2)(QUOTE AND))(GO B5))                                     00210
    ((EQUAL(CAADAR U2)(CAR U1))(GO B6)))                                       00220
B3B  (SETQ U2 (CDR U2))                                                        00230
  (GO B3A)                                                                     00240
B6  (SETQ U3 (GMATCH(CDR U1)(CDADAR U2)))                                      00250
  (COND((NULL U3)(GO B3B)))                                                    00260
  (SETQ U3 (PUTN2 NIL (CAAR U2)(CADDDAR U)))                                   00270
  (COND((NULL U3)(GO B3B))                                                     00280
    ((MEMBER U3 (QUOTE(DONE NEXT)))(RETURN U3)))                               00290
  (SETQ S U3)                                                                  00300
  (GO B3B)                                                                     00310
B5  (COND((MEMBER(CAR U1)(ATMS(CDADAR U2)))                                    00320
  (SETQ U2 (APPEND U2 (LIST(LIST(CAAR U2)(CAR(CDADAR U2)))                     00330
    (LIST(CAAR U2)(CADR(CDADAR U2))))))))                                      00340
  (GO B3B)                                                                     00350
B2  (COND((NOT(NULL S))(RETURN S)))                                            00360
EXTEND  (RETURN NIL)                                                           00370
)))))    STOP))))))                                                           00380
```

```
                                SOLVE          LISP
DEFINE(((SOLVE(LAMBDA (U)                                                    00010
(PROG(U1 U2 U3 U4 Z5 S)                                                      00020
  (COND((EQUAL(CAR U)(QUOTE INDUCTION))(GO I)))                              00030
H  (SETQ U1 (PUTN2 NIL U NIL))                                               00040
  (COND((EQUAL U1 (QUOTE DONE))(GO H4)))                                     00050
H2  (COND((EQUAL(CAAAR W)(QUOTE REL))(GO H3)))                               00060
  (SETQ W1 (APPEND W1 (CONS(CAR W) NIL)))                                    00070
  (SETQ W (CDR W))                                                           00080
  (GO H2)                                                                    00090
H3  (SETQ W (CONS(SUBST(QUOTE PNT)(QUOTE REL)(CAR W))(CDR W)))               00100
H3A  (COND((EQUAL(CADAR W)(QUOTE IMPLIES))(GO HH))                           00110
    ((EQUAL Z5 (QUOTE INDUCTION))(GO I1)))                                   00120
  (SETQ U (SCANW NIL NIL))                                                   00130
H3B  (COND((EQUAL U (QUOTE DONE))(GO H4))                                    00140
    ((EQUAL U (QUOTE NEXTV))(GO H5A))                                        00150
    ((EQUAL U (QUOTE NEXT))(GO H5)))                                         00160
HH  (SETQ U (CAR W))                                                         00170
  (COND((CONNECTIVE(CADR U))(GO H6)))                                        00180
  (SETQ U1 (GTFRM(CDR U)(QUOTE X)(CADDDAR U) NIL))                           00190
  (COND((NULL U1)(GO H8))                                                    00200
    ((EQUAL U1 (QUOTE DONE))(GO H4)))                                        00210
  (GO H5A)                                                                   00220
H8  (SETQ U1 (ALLVBLES(TERMS(CDR U)(ALLVBLES(VBLES(CDR U))))))               00230
  (SETQ S NIL)                                                               00240
H8A  (COND((NULL U1)(GO H8B)))                                               00250
  (SETQ U2 (CONS(CAR U1) NIL))                                               00260
  (SETQ U3 (CDR U1))                                                         00270
  (SETQ U1 NIL)                                                              00280
H81  (COND((NULL U3)(GO H82))                                                00290
    ((EQUAL(CAAR U2)(CAAR U3))(SETQ U2 (APPEND U2                            00300
      (CONS(CAR U3) NIL))))                                                  00310
    (T (SETQ U1 (APPEND U1 (CONS(CAR U3) NIL)))))                            00320
  (SETQ U3 (CDR U3))                                                         00330
  (GO H81)                                                                   00340
H82  (COND((NULL U2)(GO H83)))                                               00350
  (SETQ U3 (GTFRM(CAR U2)(QUOTE Y) NIL Z4B))                                 00360
  (COND((NULL U3)(GO H821))                                                  00370
    ((EQUAL U3 (QUOTE DONE))(GO H4))                                         00380
    ((EQUAL U3 (QUOTE NEXT))(GO H5A1))                                       00390
    ((EQUAL(CAAR U2)(QUOTE ISOMORPHISM))(SETQ Z4B (ADD1 Z4B))))              00400
  (SETQ Z5 (QUOTE GEN))                                                      00410
H821  (SETQ U2 (CDR U2))                                                     00420
  (GO H82)                                                                   00430
H83  (SETQ Z4B (ADD1 Z4B))                                                   00440
  (GO H8A)                                                                   00450
H8B  (COND((NULL Z5)(GO H9))                                                 00460
    ((NOT(NULL U))(SETQ Z5 NIL)))                                            00470
  (SETQ U3 (SCANW NIL Z5))                                                   00480
  (SETQ Z5 NIL)                                                              00490
  (COND((NULL U3)(GO H9)))                                                   00500
  (SETQ U U3)                                                                00510
  (GO H3B)                                                                   00520
H9  (COND((NULL U)(GO H10A))                                                 00530
    ((EQ S N)(GO H10)))                                                      00540
  (SETQ S N)                                                                 00550
  (SETQ U2 (ALLVBLES(VBLES(CDR U))))                                         00560
  (COND((NULL U2)(GO H10)))                                                  00570
H91  (SETQ U1 (APPEND U1 (ARGOF1(CAR U2) VV1)))                              00580
  (SETQ U2 (CDR U2))                                                         00590
  (COND((NOT(NULL U2))(GO H91)))                                             00600
  (GO H8A)                                                                   00610
H6  (COND((EQUAL(CADR U)(QUOTE IMPLIES))(GO H6A))                            00620
    ((EQUAL(CADR U)(QUOTE DEFER))(GO H6E)))                                  00630
  (GO H8)                                                                    00640
H6A  (SETQ U1 (ALLVBLES(VBLES(CADDR U))))                                    00650
  (SETQ U1 (RPLC1 U1))                                                       00660
  (COND((NOT(NULL U1))(SETQ U (CONS(CAR U)(GENSUBST                          00670
      (CAR U1)(CDR U1)(CDR U))))))                                           00680
  (SETQ U1 (PUTON1(CONS Z4A Z4B)(CADDR U)))                                  00690
  (SETQ Z4B (ADD1 Z4B))                                                      00700
  (COND((EQUAL U1 (QUOTE DONE))(GO H4))                                      00710
    ((EQUAL U1 (QUOTE NEXT))(GO H5A)))                                       00720
  (SETQ U1 (PUTN2 NIL (CADDDR U)(CADDDAR U)))                                00730
H6A1  (COND((NULL U1)(GO H8))                                                00740
    ((EQUAL U1 (QUOTE DONE))(GO H4)))                                        00750
  (GO H5A)                                                                   00760
H6E  (SETQ U1 (PUTN2 NIL (CADDR U)(CADDDAR U)))                              00770
  (GO H6A1)                                                                  00780
H10  (COND((EQUAL(CADR U)(QUOTE EXISTS))(GO H6B)))                           00790
H10C  (SETQ U NIL)                                                           00800
  (SETQ U1 CL)                                                               00810
  (GO H8A)                                                                   00820
```

```
H10A  (COND((EQUAL(CADAR W)(QUOTE EXISTS))(GO H6B))                    00830
    ((EQUAL(CADAR W)(QUOTE FEQUAL))(GO H6D)))                          00840
H10B  (PRINT(QUOTE(DRASTIC MEASURES NEEDED)))                          00850
  (PRINT(QUOTE(TYPE   YES   IF POINTER SHOULD BE MOVED DOWN)))         00860
  (SETQ U (RDFLX()))                                                   00870
  (COND((EQUAL U (QUOTE YES))(GO H5)))                                 00880
  (GO H4F)                                                             00890
H6B  (SETQ U1 (SOLVEX(CAR W)))                                         00900
H6B1  (COND((NULL U1)(GO H6BA))                                        00910
    ((EQUAL U1 (QUOTE NEXT))(GO H5))                                   00920
    ((EQUAL U1 N)(GO H5A))                                             00930
    ((EQUAL U1 (QUOTE DONE))(GO H4)))                                  00940
H6BA  (COND((NOT(NULL U))(GO H10C)))                                   00950
  (SETQ U1 (SOLVX(CAR W)))                                             00960
  (COND((NOT(NULL U1))(GO H6B1)))                                      00970
  (PRINT(QUOTE EXISTS))                                                00980
  (GO H10B)                                                            00990
H6D  (SETQ U1 (HOMOMF(CDAR W)))                                        01000
  (COND((EQUAL U1 (QUOTE NEXT))(GO H5))                                01010
    ((NOT(EQUAL U1 (QUOTE DONE)))(GO H10B)))                           01020
H4  (PRINT VV1)(PRINT VV2)(PRINT W1)(PRINT W)                          01030
  (RETURN T)                                                           01040
H51  (SETQ Z4B (ADD1 Z4B))                                             01050
H5  (COND((NULL (NEXT NIL))(GO H5E)))                                  01060
  (GO HH)                                                              01070
H5A1  (SETQ Z4B (ADD1 Z4B))                                            01080
H5A  (COND((NOT(NULL(NEXT NIL)))(GO H3A)))                             01090
H5E  (PRINT(QUOTE(PANIC H5E SOLVE)))                                   01100
H4F  (PRINT VV1)(PRINT VV2)(PRINT W1)(PRINT W)                         01110
  (RETURN F)                                                           01120
I  (SETQ Z5 (CAR U))                                                   01130
  (SETQ U (CDR U))                                                     01140
  (GO H)                                                               01150
I1  (SETQ Z5 NIL)                                                      01160
  (GO HH)                                                              01170
)))))) STOP)))))))                                                     01180
```

```
                              SCANW          LISP
DEFINE(((SCANW(LAMBDA(L LL)                                                    00010
(PROG(S S1 U U1 U2 U3 U4 U5 U6 U7 U8 FLG)                                       00020
A1  (SETQ S NIL)                                                               00030
  (SETQ FLG NIL)                                                               00040
  (SETQ U1 (ONLY(APPEND W1 W)(QUOTE(VERA2))))                                  00050
A1A  (COND((NULL U1)(GO A2))                                                   00060
     ((MEMBER(CADDDAAR U1) Z6)(GO AA))                                         00070
     ((MEMBER(QUOTE HEAD)(CADAAR U1))(SETQ FLG (CONS(SIXTH                     00080
       (CAAR U1)) FLG))))                                                      00090
AA  (SETQ U1 (CDR U1))                                                         00100
  (GO A1A)                                                                     00110
A2  (COND((NULL W)(SETQ U1 NIL))                                               00120
     ((NULL LL)(SETQ U1 (CONS(CAR W) NIL)))                                    00130
     ((EQUAL LL N)(SETQ U1 (CDR W)))                                           00140
     (T (SETQ U1 W)))                                                          00150
A2A  (COND((NULL U1)(GO C)))                                                   00160
  (SETQ U2 (CAR U1))  (SETQ U1 (CDR U1))                                       00170
  (COND((NOLOOK(CADDDAR U2))(GO A2A))                                          00180
    ((MEMBER(CADR U2)(QUOTE(IMPLIES FEQUAL DEFER OR)))(GO A2A)))               00190
  (SETQ U (ROFV(FIFTH(CAR U2)) FLG))                                           00200
  (SETQ U4 NIL)                                                                00210
  (CHNGW U2)                                                                   00220
A2B  (COND((NULL U)(GO A3)))                                                   00230
  (SETQ U3 (CAR U))  (SETQ U (CDR U))                                          00240
  (COND((EQUAL(CADR U3)(QUOTE IMPLIES))(GO B4))                                00250
    ((EQUAL(CADR U2)(QUOTE EXISTS))(GO A2B))                                   00260
    ((MEMBER(CONS(CADDDAR U2)(CDAR U3)) LON)(GO A2B))                          00270
    ((EQUAL(CADR U3)(QUOTE EQUAL))(GO B3))                                     00280
    ((EQUAL(CADR U3)(QUOTE EQUAL2))(GO B2))                                    00290
    ((EQUAL(CADR U3)(QUOTE ASSOC))(GO B6)))                                    00300
  (GO A2B)                                                                     00310
A3  (COND((NULL U4)(GO A2A))                                                   00320
    ((NULL S)(GO A31))                                                         00330
    ((NOLOOK(CADDDAR U2))(GO A2A)))                                            00340
A31  (SETQ U5 (CAAR U4))                                                       00350
  (SETQ U6 (CDAR U4))                                                          00360
  (SETQ U7 (CDR U4))                                                           00370
  (SETQ U4 NIL)                                                                00380
A3A  (COND((NULL U7)(GO A3B))                                                  00390
    ((EQUAL U5 (CAAR U7))(SETQ U6 (APPEND U6 (CDAR U7))))                      00400
    (T (SETQ U4 (APPEND U4 (CONS(CAR U7) NIL)))))                              00410
  (SETQ U7 (CDR U7))                                                           00420
  (GO A3A)                                                                     00430
A3B  (COND((MEMBER(CADR U2)(QUOTE(EQUAL EQUAL2)))(SETQ U (CADDR U2)))         00440
    (T (SETQ U (CDR U2))))                                                     00450
  (SETQ U6 (SUPXEC U6 U))                                                      00460
A3D  (COND((NULL U6)(GO A3))                                                   00470
    ((MEMBER(CADR U2)(QUOTE(EQUAL EQUAL2)))                                    00480
      (SETQ U (LIST(CADR  U2)(CAR U6)(CADDDR U2))))                            00490
    (T (SETQ U (CAR U6))))                                                     00500
  (SETQ U6 (CDR U6))                                                           00510
  (SETQ U3 (PUTN2(COND((MEMBER(CAAR U2)(QUOTE(REL1 PNT)))                      00520
      NIL)(T (CAAR U2))) U (CADDDAR U2)))                                      00530
  (COND((NULL U3)(GO A3D))                                                     00540
    ((EQUAL U3 (QUOTE DONE))(RETURN U3))                                       00550
    ((EQUAL U3 (QUOTE NEXT))(GO A3E)))                                         00560
A3D1  (SETQ S N)                                                               00570
  (GO A3D)                                                                     00580
A3E  (COND((NULL LL)(RETURN(QUOTE NEXTV))))                                    00590
  (SETQ S1 U3)                                                                 00600
  (GO A3D1)                                                                    00610
B2  (SETQ U5 (ALLSUBSTS(CADDR U3)(CADDDR U3)                                   00620
    (CDR U2)(CDAR U3)))                                                        00630
  (COND((NULL U5)(GO B3)))                                                     00640
  (SETQ U4 (APPEND U4 (CONS U5 NIL)))                                          00650
B3  (SETQ U5 (ALLSUBSTS(CADDDR U3)(CADDR U3)                                   00660
    (CDR U2)(CDAR U3)))                                                        00670
B3A  (COND((NOT(NULL U5))(SETQ U4 (APPEND U4 (CONS U5 NIL)))))                 00680
  (GO A2B)                                                                     00690
B6  (COND((NOT(MEMBER(QUOTE *PROD)(ALLVBLES(ATMS(CDR U2)))))                   00700
      (GO A2B)))                                                               00710
    (SETQ U5 (ASSOCM(CADDR U3)(CDR U2)(CDAR U3)))                              00720
  (GO B3A)                                                                     00730
B4  (SETQ U5 (ALLMATCH1(CADDDR U3)(CDR U2)))                                   00740
B41  (COND((NULL U5)(GO B8))                                                   00750
    ((NULL S)(GO B41A))                                                        00760
    ((NOLOOK (CADDDAR U2))(GO A2A)))                                           00770
B41A  (SETQ U6 (PUTN2(COND((MEMBER(CAAR U2)(QUOTE(REL1 PNT)))                  00780
      NIL)(T (CAAR U2)))(GENSUBST(CDAR U5)(CAAR U5)                            00790
      (CADDR U3))(CADDDAR U2)))                                                00800
  (COND((NULL U6)(GO B4A))                                                     00810
    ((EQUAL U6 (QUOTE DONE))(RETURN U6))                                       00820
    ((EQUAL U6 (QUOTE NEXT))(GO B4B)))                                         00830
```

```
B41B  (SETQ S N)                                                                00840
B42   (COND((EQUAL(CAAR U2)(QUOTE REL))(GO B4C))                                00850
      ((EQUAL(CAAR U2)(QUOTE PNT))(SETQ S1 (QUOTE NEXT))))                      00860
B43   (SETQ LON (CONS(CONS(CADDDAR U2)(CDAR U3)) LON))                          00870
B43A  (SETQ U5 (CDR U5))                                                        00880
  (GO B41)                                                                      00890
B4A   (COND((ON(GENSUBST(CDAR U5)(CAAR U5)(CADDR U3))                          00900
      (ROFW(CADDDAR U2)))(GO B42)))                                             00910
  (GO B43A)                                                                     00920
B4B   (COND((NULL LL)(RETURN(QUOTE NEXTV))))                                   00930
  (SETQ S1 U6)                                                                  00940
  (GO B41B)                                                                     00950
B4C   (SETQ U6 (GFIND(CADDDAR U2)))                                            00960
  (COND((EQUAL(CAAR U6)(QUOTE REL))(SETQ W (SUBST                               00970
    (SUBST(QUOTE REL1)(QUOTE REL) U6) U6 W))))                                  00980
  (GO B43)                                                                      00990
B8    (COND((MEMBER(CONS(CADDDAR U2)(CDAR U3)) LON)(GO A2B)))                   01000
  (SETQ U5 (IMPMATCH(CADDDR U3)(CDR U2)))                                       01010
A10   (COND((NULL U5)(GO A2B))                                                 01020
      ((NULL S)(GO B8B))                                                        01030
      ((NOLOOK(CADDDAR U2))(GO A2A)))                                           01040
B8B   (SETQ U6 (GENSUBST(CDAR U5)(CAAR U5)(CADDR U3)))                         01050
  (COND((ON U6 (APPEND VV1 VV2))(GO B8A))                                       01060
      ((EQUAL(CAR U6)(QUOTE AND))(GO A10C)))                                    01070
A10B  (SETQ U7 (ON2AVER U6))                                                    01080
  (COND((NOT(NULL U7))(GO A14))                                                 01090
      ((EQUAL(CAAR U2)(QUOTE A2))(GO A10A))                                     01100
      ((NULL L)(GO A10F))                                                       01110
      ((EQUAL L U6)(GO A10G)))                                                  01120
A10A  (SETQ U5 (CDR U5))                                                        01130
  (GO A10)                                                                      01140
A10G  (COND((NOT(EQUAL(CADR U2)(QUOTE EXIST)))(GO A10A)))                      01150
  (SETQ S (SCNX S (CADR L)(CAR U5)(CADDDR U3) U2))                              01160
  (COND((EQUAL S (QUOTE DONE))(RETURN S))                                       01170
      ((ATOM S)(GO A10A)))                                                      01180
  (SETQ L (CDR S))  (SETQ S (CAR S))                                            01190
  (GO A10A)                                                                     01200
A10F  (SETQ U6 (PUTON2(LIST(QUOTE A2)(LIST(QUOTE HEAD))                         01210
      (LIST(QUOTE NONE)) Z4 N (CAAR U3)) U6 NIL))                               01220
  (COND((NOT(NULL U6))(SETQ S N)))                                              01230
  (GO A10A)                                                                     01240
A10C  (SETQ U7 (ANDARGS U6))                                                    01250
A10D  (COND((NULL U7)(GO B8A))                                                  01260
      ((EQUAL(CAAR U7)(QUOTE AND))(GO A10E))                                    01270
      ((ON(CAR U7)(APPEND VV1 VV2))(GO A10E)))                                  01280
  (GO A10B)                                                                     01290
A10E  (SETQ U7 (CDR U7))                                                        01300
  (GO A10D)                                                                     01310
A14   (COND((MEMBER(QUOTE HEAD) U7)(GO A14A))                                  01320
      ((EQUAL(CAAR U2)(QUOTE A2))(GO A10A)))                                    01330
A14A  (SETQ U8 (CONS Z4A U8))                                                   01340
  (SETQ U7 (PUTON1(CONS Z4A (CDAR U3)) U6))                                     01350
  (COND((NULL U7)(GO B8B))                                                      01360
      ((MEMBER(CAR U6)(QUOTE(AND EXISTS)))                                      01370
        (SETQ U (APPEND U (ROFV(CAR U8) NIL))))                                 01380
      ((MEMBER(CAR U6)(QUOTE(EQUAL2 EQUAL IMPLIES)))                            01390
        (SETQ U (APPEND U (CONS(CONS(CONS(CAR U8)                               01400
        (CDAR U3)) U6)NIL)))))                                                  01410
  (COND((EQUAL U7 (QUOTE DONE))(RETURN U7))                                     01420
      ((EQUAL U7 (QUOTE NEXT))(SETQ S1 U7)))                                    01430
  (SETQ S N)                                                                    01440
  (COND((EQ(CDR U8) N)(GO B8B)))                                                01450
B8A   (SETQ U6 (GENSUBST(CDAR U5)(CAAR U5)(CADDDR U3)))                         01460
  (SETQ U8 N)                                                                   01470
  (GO A14A)                                                                     01480
B8B   (SETQ U8 NIL)                                                             01490
  (GO A10A)                                                                     01500
C   (SETQ Z6 (APPEND FLG Z6))                                                   01510
  (COND((NULL LL)(GO D))                                                        01520
      ((NULL S)(RETURN S1))                                                     01530
      ((NULL S1)(SETQ LL (QUOTE GEN)))                                          01540
      ((EQUAL S1 (QUOTE NEXT))(GO C1)))                                         01550
  (GO A1)                                                                       01560
C1   (SETQ U1 (NEXT NIL))                                                       01570
  (COND((NULL U1)(GO C2)))                                                      01580
  (SETQ S1 N)                                                                   01590
  (SETQ LL (QUOTE GEN))                                                         01600
  (GO A1)                                                                       01610
C2   (PRINT(QUOTE(PANIC C2 SCANW)))                                            01620
  (RETURN(QUOTE DONE))                                                          01630
D   (COND((NOT(NULL S1))(RETURN (SUBST(QUOTE NEXTV)(QUOTE NEXT) S1)))          01640
      ((NULL S)(SETQ LL N)))                                                    01650
  (GO A1)  )))))) STOP))))))                                                    01660
```

```
                        PUTON1        LISP
DEFINE(((PUTON1(LAMBDA(L U)(PROG(U1 U3 U4)                                    00010
  (COND((MEMBER(CAR U)(QUOTE(IMPLIES EQUAL EQUAL2 ASSOC)))                    00020
      (GO C))                                                                 00030
     ((EQUAL(CAR U)(QUOTE AND))(GO B2))                                       00040
     ((EQUAL(CAR U)(QUOTE IMPLIES2))(GO B))                                   00050
     ((ON U VV1)(RETURN NIL)))                                                00060
  (SETQ U1 N)                                                                 00070
  (SETQ Z4A (ADD1 Z4A))                                                       00080
  (SETQ VV1 (APPEND VV1 (CONS(CONS L U) NIL)))                                00090
  (COND((EQUAL(CAR U)(QUOTE EXISTS))(GO B1))                                  00100
    ((EQUAL(CAR U)(QUOTE GROUP))(SETQ Q (ALLVBLES(CONS                        00110
      (CADR U) Q))))                                                          00120
    ((MEMBER(CAR U)(QUOTE(HOMOMORPHISM ISOMORPHISM                            00130
      EPIMORPHISM MONOMORPHISM)))(SETQ QL1                                    00140
      (GTFRM(CONS(QUOTE HMPRP)(CDR U))                                        00150
      (QUOTE XX) NIL QL1)))                                                   00160
    ((EQUAL(CAR U)(QUOTE FACTORGROUP))(SETQ QL2 (GTFRM                        00170
      (CONS(QUOTE FGPRP)(CDR U))(QUOTE XX) NIL QL2))))                        00180
  (COND((NULL QM5)(GO A2))                                                    00190
    ((AND(EQUAL(CAR U)(QUOTE MEMBER))(ATOM(CADR U)))                          00200
      (SETQ QM2 (CONS(CDR U) QM2))))                                          00210
  (MODELMAKE U)                                                               00220
A2  (COND((NOT(ON U (APPEND W1 W)))(RETURN U1)))                              00230
A1  (SETQ U3 (VERIFY U))                                                      00240
  (COND((NULL U3)(RETURN U1)))                                                00250
  (RETURN U3)                                                                 00260
C  (COND((ON U VV2)(RETURN NIL)))                                            00270
  (SETQ U1 N)                                                                 00280
  (SETQ Z4A (ADD1 Z4A))                                                       00290
  (SETQ VV2 (APPEND VV2 (CONS(CONS L U) NIL)))                                00300
  (COND((EQUAL(CAR U)(QUOTE ASSOC))(SETQ Q (ALLVBLES                          00310
      (CONS(CADR U) Q)))))                                                    00320
  (COND((ON U (APPEND W1 W))(GO A1))                                          00330
    ((EQUAL(CAR U)(QUOTE EQUAL))(SETQ U (CONS(QUOTE EQUAL2                    00340
      )(CDR U))))                                                             00350
    ((EQUAL(CAR U)(QUOTE EQUAL2))(SETQ U (CONS(QUOTE EQUAL)                   00360
      (CDR U)))))                                                             00370
  (GO A2)                                                                     00380
B1  (COND((NOT(ON U (APPEND W1 W)))(GO B1A)))                                 00390
  (SETQ U3 (VERIFY U))                                                        00400
  (COND((EQUAL U3 (QUOTE DONE))(RETURN U3))                                   00410
    ((EQUAL U3 (QUOTE NEXT))(SETQ U1 U3)))                                    00420
B1A  (COND((ON(CADR U) Z3)(GO B1B)))                                          00430
  (PRINT(QUOTE(EXISTS B1A PUTON1)))                                           00440
  (RETURN (QUOTE DONE))                                                       00450
B1B  (SETQ U3 (GENSYM))                                                       00460
  (SETQ U4 (SUBST U3 (CADR U)(CADDR U)))                                      00470
  (COND((NULL(GENVBLE(CADDR U)))(SETQ Z3 (CONS                                00480
      (CONS(QUOTE RES) U3) Z3)))                                              00490
    (T (SETQ Z3 (CONS(CONS(QUOTE GEN) U3) Z3))))                              00500
  (SETQ U3 (PUTON1(CONS Z4A (CDR L)) U4))                                     00510
  (GO B2A)                                                                    00520
B  (SETQ U (LIST N (LIST(QUOTE IMPLIES)(SUBST(QUOTE EQUAL)                    00530
    (QUOTE SEQUAL)(CADDR U))(CADR U))(LIST(QUOTE IMPLIES)                     00540
    (CADR U)(SUBST(QUOTE EQUAL2)(QUOTE SEQUAL)(CADDR U)))))                   00550
B2  (COND((NOT(EQUAL(CDR L) N))(GO B21))                                      00560
    ((EQUAL(CAADR U)(QUOTE AND))(SETQ U1 (PUTON1 L (CADR U))))                00570
    (T (SETQ U1 (PUTON1(CONS(CAR L) Z4B)(CADR U)))))                          00580
  (SETQ Z4B (ADD1 Z4B))                                                       00590
  (GO B22)                                                                    00600
B21  (SETQ U1 (PUTON1 L (CADR U)))                                            00610
B22  (COND((EQUAL U1 (QUOTE DONE))(RETURN U1))                                00620
    ((NOT(EQUAL(CDR L) N))(SETQ L (CONS Z4A (CDR L))))                        00630
    ((EQUAL(CAADDR U)(QUOTE AND))(SETQ L (CONS Z4A N)))                       00640
    (T (SETQ L (CONS Z4A Z4B))))                                              00650
  (SETQ U3 (PUTON1 L (CADDR U)))                                              00660
B2A  (COND((OR(NULL U1)(MEMBER U3 (QUOTE(NEXT DONE))))(RETURN U3)))          00670
  (RETURN U1)                                                                 00680
)))))) STOP))))))                                                             00690
```

```
                          PUTON2        LISP
DEFINE(((PUTN2(LAMBDA(L U M)(PROG(U1)                                          00010
  (COND((NULL L)(SETQ L (QUOTE REL))))                                         00020
  (COND((NULL M)(SETQ U1 (CONS(QUOTE HEAD) NIL)))                              00030
    ((ATOM M)(SETQ U1 (CONS M NIL)))                                           00040
    (T (SETQ U1 (CONS(CAR M) NIL))))                                           00050
  (RETURN(PUTON2(LIST L U1 (CONS(QUOTE NONE) NIL) Z4 N)                        00060
    U M)) )))))                                                                00070
DEFINE(((PUTON2(LAMBDA(L U M)(PROG()                                           00080
  (COND((ON U (APPEND W1 W))(RETURN(ON2 L U M)))                               00090
    ((EQUAL(CAR L)(QUOTE A2))(GO A)))                                          00100
B0  (RETURN(PUT2 L U M))                                                       00110
B1  (COND((MEMBER(CDR U) QM2)(GO B0))                                          00120
    ((MEMBER(CDR U) QM6)(GO B3))                                               00130
    ((NULL(CHECKMODEL(CDR U)))(GO B2)))                                        00140
  (GO B0)                                                                      00150
B2  (SETQ QM6 (CONS(CDR U) QM6))                                               00160
B3  (COND((ATOM M)(RETURN NIL)))                                               00170
  (RETURN (QUOTE REJECT))                                                      00180
A  (COND((MEMBER(CAR U)(QUOTE(IMPLIES EXISTS IMPLIES2 FEQUAL)))               00190
    (RETURN NIL))  ((NULL QM5)(GO B0))                                         00200
    ((AND(EQUAL(CAR U)(QUOTE MEMBER))(ATOM(CADR U)))(GO B1)))                  00210
  (GO B0) )))))                                                                00220
DEFINE(((ON2(LAMBDA(L U M)(PROG(U3)                                            00230
  (COND((NULL M)(GO A1D))                                                      00240
    ((ATOM M)(GO A1A)))                                                        00250
  (SETQ U3 (ADDANTEC(CAR M) U))                                                00260
  (COND((NULL U3)(GO A1C))                                                     00270
    ((NULL(CDR M))(RETURN U3))                                                 00280
    ((EQUAL U3 (QUOTE B))(GO A1C)))                                            00290
  (SETQ U3 (ADDCONSQ(CONS(CADR M) U3)(CAR M)))                                 00300
  (RETURN N)                                                                   00310
A1A  (SETQ U3 (ADDANTEC M U))                                                  00320
  (COND((NULL U3)(RETURN NIL))                                                 00330
    ((EQUAL U3 (QUOTE B))(GO A1F)))                                            00340
  (SETQ U3 (ADDCONSQ U3 M))                                                    00350
  (RETURN N)                                                                   00360
A1C  (COND((NULL(CDR M))(RETURN NIL)))                                         00370
  (SETQ U3 (ADDCONSQ(CADR M)(CAR M)))                                          00380
  (RETURN N)                                                                   00390
A1D  (COND((NOT(EQUAL(CAR L)(QUOTE A2)))(RETURN NIL)))                        00400
  (SETQ U3 (CAR(GFIND U)))                                                     00410
  (COND((NOT(MEMBER(CAR U3)(QUOTE(A2 VERA2))))(RETURN NIL))                   00420
    ((MEMBER(QUOTE HEAD)(CADR U3))(RETURN NIL)))                               00430
  (SETQ M (SUBST(CONS(QUOTE HEAD)(CADR U3))(CADR U3) U3))                      00440
  (SETQ M (APPEND M (CONS(SIXTH L) NIL)))                                      00450
  (COND((OR(NULL W)(LESSP(CADDDR U3)(CADDDAAR W)))                             00460
      (SETQ W1 (SUBST M U3 W1)))                                               00470
    (T (SETQ W (SUBST M U3 W))))                                               00480
  (RETURN NIL)                                                                 00490
A1F  (SETQ U3 (VERIFY(CDR(GFIND M))))                                          00500
  (COND((NULL U3)(RETURN N)))                                                  00510
  (RETURN U3) )))))                                                            00520
COMPILE((PUTN2 PUTON2 ON2))   STOP))))                                         00530
```

```
                          PUT2        LISP
DEFINE(((PUT2(LAMBDA(L U M)(PROG(Z U1 U2 U3 U4 L1)                             00010
  (SETQ U1 N)                                                                  00020
  (SETQ Z Z4)                                                                  00030
  (SETQ Z4 (ADD1 Z4))                                                          00040
  (COND((MEMBER(CAR U)(QUOTE(AND OR IMPLIES2)))(GO B4)))                       00050
B5  (SETQ W (APPEND W (CONS (CONS L U) NIL)))                                  00060
  (COND((NULL M)(GO A2))                                                       00070
    ((ATOM M)(GO B1))                                                          00080
    ((NULL(CDR M))(GO A2A)))                                                   00090
  (SETQ U3 (ADDCONSQ(CDR M)(CAR M)))                                           00100
  (GO A2)                                                                      00110
A2A  (SETQ U1 (QUOTE A1))                                                      00120
  (GO A2)                                                                      00130
B1  (SETQ U3 (ADDCONSQ Z M))                                                   00140
A2  (COND((ON U (APPEND VV1 VV2))(GO A3))                                      00150
    ((NOT(NULL QF1))(GO C1)))                                                  00160
A2C  (COND((MEMBER(CAR U)(QUOTE(IMPLIES EQUAL EQUAL2)))(GO A6))               00170
    ((EQUAL(CAR U)(QUOTE IMPLIES2))(GO I))                                     00180
    ((AND(NULL CX)(EQUAL(CAR U)(QUOTE EXIST)))(GO E))                          00190
    ((EQUAL(CAR U)(QUOTE OR))(GO D))                                           00200
    ((EQUAL(CAR U)(QUOTE AND))(GO A7)))                                        00210
```

```
A28  (SETQ U4 (ALLVBLES(ATMS U)))                                                00220
  (SETQ U2 (INVM U U4))                                                          00230
  (COND((NULL U2)(RETURN(HFM U U4 U1 L Z))))                                     00240
  (SETQ W (SUBST(CONS N (CDR L)) L W))                                           00250
  (SETQ U3 (PUTN2(CAR L) U2 Z))                                                  00260
  (GO A12A)                                                                      00270
B4  (SETQ L1 (CAR L))                                                            00280
  (COND((NOT(EQUAL L1 (QUOTE A2)))(SETQ L (CONS N (CDR L)))))                    00290
  (GO B5)                                                                        00300
A3  (SETQ U3 (VERIFY U))                                                         00310
  (COND((EQUAL U1 (QUOTE A))(SETQ U1 (QUOTE B)))                                 00320
    ((EQUAL U3 (QUOTE DONE))(RETURN U3))                                         00330
    ((EQUAL U3 (QUOTE NEXT))(GO A9A)))                                           00340
A9B  (COND((EQUAL(CAR U)(QUOTE AND))(GO A10)))                                   00350
A8  (RETURN U1)                                                                  00360
A6  (COND((EQUAL(CAR U)(QUOTE EQUAL))(GO A6A))                                   00370
    ((EQUAL(CAR U)(QUOTE EQUAL2))(GO A6B)))                                      00380
A6C  (COND((EQUAL(CADR U)(CADDR U))(GO A3)))                                     00390
  (GO A2B)                                                                       00400
A6A  (COND((ON(CONS(QUOTE EQUAL2)(CDR U)) VV2)(GO A3)))                          00410
  (GO A6C)                                                                       00420
A6B  (COND((ON(CONS(QUOTE EQUAL)(CDR U)) VV2)(GO A3)))                           00430
  (GO A6C)                                                                       00440
A7  (SETQ U2 Z4)                                                                 00450
  (COND((AND(EQUAL(CAADDR U)(QUOTE MEMBER))(ATOM(CAR(CDADDR U))))(GO A7A         00460
)))                                                                              00470
A7B  (SETQ U3 (PUTN2 L1 (CADR U)(CONS Z NIL)))                                   00480
  (COND((NULL U3)(GO A12))                                                       00490
    ((MEMBER U3 (QUOTE(NILL A)))(GO A12))                                        00500
  ((EQUAL U3 (QUOTE B))(GO A13A))                                                00510
   ((EQUAL U3 (QUOTE NEXTV))(GO A13B))                                           00520
    ((EQUAL U3 (QUOTE DONE))(RETURN U3))                                         00530
    ((EQUAL U3 (QUOTE NEXT))(SETQ U1 U3))                                        00540
    ((EQUAL U3 (QUOTE REJECT))(RETURN NIL))                                      00550
    (T (SETQ U2 U3)))                                                            00560
A12  (SETQ U3 (PUTN2 L1 (CADDR U)(CONS Z (CONS U2 Z4))))                         00570
A12A  (COND((MEMBER U3 (QUOTE(DONE NEXT)))(RETURN U3)))                          00580
  (GO A8)                                                                        00590
A7A  (SETQ U (LIST(CAR U)(CADDR U)(CADR U)))                                     00600
  (COND((NULL QM5)(GO A7B))                                                      00610
    ((AND(EQUAL(CAADDR U)(QUOTE MEMBER))(ATOM(CAR(CDADDR U))))                   00620
      (SETQ U (NOWWHICH U))))                                                    00630
  (GO A7B)                                                                       00640
A10  (SETQ U4 (ANDARGS U))                                                       00650
A16  (COND((NULL U4)(GO A8))                                                     00660
    ((ON(CAR U4)(APPEND W1 W))(GO A15)))                                         00670
A17  (SETQ U4 (CDR U4))                                                          00680
  (GO A16)                                                                       00690
A15  (SETQ U3 (VERIFY(CAR U4)))                                                  00700
    (COND((EQUAL U3 (QUOTE DONE))(RETURN U3))                                    00710
    ((EQUAL U3 (QUOTE NEXT))(GO A15A)))                                          00720
  (GO A17)                                                                       00730
A15A  (COND((EQUAL U1 (QUOTE B))(SETQ U1 (QUOTE NEXTV)))                         00740
    (T (SETQ U1 (QUOTE NEXT))))                                                  00750
  (GO A17)                                                                       00760
A9A  (COND((EQUAL U1 (QUOTE B))(SETQ U1 (QUOTE NEXTV)))                          00770
    (T (SETQ U1 (QUOTE NEXT))))                                                  00780
  (GO A9B)                                                                       00790
A13B  (SETQ U1 (QUOTE NEXT))                                                     00800
A13A  (SETQ U3 (PUTN2 L1 (CADDR U) Z))                                           00810
  (GO A12A)                                                                      00820
C1  (COND((AND(EQUAL(CAR L)(QUOTE REL))(ON U QF1))                               00830
    (RETURN(QUOTE DONE))))                                                       00840
  (GO A2C)                                                                       00850
D  (SETQ U3 (PUTN2 L1 (CADR U) Z))                                               00860
  (COND((EQUAL U3 (QUOTE NEXT))(SETQ U1 U3))                                     00870
    ((EQUAL U3 (QUOTE DONE))(RETURN U3)))                                        00880
  (GO A13A)                                                                      00890
I  (RETURN(PUTN2 L1 (LIST(QUOTE AND)(LIST(QUOTE IMPLIES)                         00900
    (SUBST(QUOTE EQUAL2)(QUOTE SEQUAL)(CADDR U))(CADR U))                        00910
    (LIST(QUOTE IMPLIES)(CADR U)(SUBST(QUOTE EQUAL)                              00920
    (QUOTE SEQUAL)(CADDR U)))) Z))                                               00930
E  (SETQ L1 (SLVX U))                                                            00940
  (COND((EQ L1 N)(RETURN(QUOTE DONE)))                                           00950
  )(GO A2B)                                                                      00960
)))))) COMPILE((PUT2))                                                           00970
STOP)))))                                                                        00980
```

```
                              SOLVEX        LISP
DEFINE((SOLVEX(LAMBDA(U)                                                    00010
(PROG(Z U1 S U2 U3 U4 U5 U6 QF2 L U7 FU)                                    00020
 (COND((NOT(ON(CADDR U) Z3))(RETURN NIL)))                                  00030
  (SETQ U7 (GENSYM))                                                        00040
  (SETQ Z3 (APPEND Z3 (CONS(CONS N U7) NIL)))                               00050
  (SETQ FU (CDR U))                                                         00060
  (SETQ U (SUBST U7 (CADDR U) U))                                           00070
  (SETQ Z Z4)                                                               00080
A1  (SETQ U1 (CADDDR U))                                                    00090
  (SETQ L N)                                                                00100
  (SETQ U2 (QUOTE(EQUAL EQUAL2)))                                           00110
  (COND((MEMBER(CAR U1) U2)(GO A2))                                         00120
    ((EQUAL(CAR U1)(QUOTE AND))(SETQ U1 (CDR U1)))                          00130
    (T (RETURN NIL)))                                                       00140
  (COND((MEMBER(CAAR U1) U2)(GO A1C))                                       00150
    ((EQUAL(CAAR U1)(QUOTE MEMBER))(GO A1B)))                               00160
  (RETURN NIL)                                                              00170
A1B  (COND((EQUAL(CADAR U1) U7)(SETQ L (CAR U1)))                           00180
    (T (RETURN NIL)))                                                       00190
  (COND((MEMBER(CAADR U1) U2)(SETQ U1 (CADR U1)))                           00200
    (T (RETURN NIL)))                                                       00210
  (GO A1D)                                                                  00220
A1C  (COND((EQUAL(CAADR U1)(QUOTE MEMBER))(SETQ L (CADR U1)))               00230
    (T (RETURN NIL)))                                                       00240
  (COND((EQUAL(CADR L) U7)(SETQ U1 (CAR U1)))                               00250
    (T (RETURN NIL)))                                                       00260
A1D  (SETQ S (CDDR L))                                                      00270
A2  (COND((MEMBER U7(VBLES(CADR U1)))(GO A2A))                              00280
    ((MEMBER U7 (VBLES(CADDR U1)))(GO A2B)))                                00290
  (RETURN NIL)                                                              00300
A2A(COND((MEMBER U7 (VBLES(CADDR U1)))(RETURN NIL)))                        00310
  (GO A3)                                                                   00320
A2B  (SETQ U1 (CONS(CAR U1)(CONS(CADDR U1)(CONS(CADR U1) NIL))))            00330
A3  (SETQ U2 (PUTN2 NIL (CONS(QUOTE EXIST)(CONS U7                          00340
  (CONS U1 NIL))) NIL))                                                     00350
  (COND((EQUAL U2 (QUOTE DONE))(GO A6E)))                                   00360
  (SETQ U2 (SCANW L (QUOTE GEN)))                                           00370
  (COND((EQUAL U2 (QUOTE NEXT))(GO E))                                      00380
    ((EQUAL U2 (QUOTE DONE))(GO A4)))                                       00390
  (GO E1)                                                                   00400
A4  (COND((NULL W1)(GO A4A))                                                00410
    ((EQUAL(CAAAR W1)(QUOTE VER))(RETURN U2)))                              00420
  (GO A6E)                                                                  00430
A4A  (COND((EQUAL(CAAAR W)(QUOTE VER))(RETURN U2)))                         00440
A6E  (COND((NULL S)(GO E2)))                                                00450
E3  (SETQ W (MASK Z))                                                       00460
  (SETQ QF2 NIL)                                                            00470
  (SETQ U1 (QUOTE MEMBER))                                                  00480
  (SETQ U3 (PUTN2 NIL (CONS U1 (CONS CX1 S))                                00490
    (CADDDAR U)))                                                           00500
  (COND((NULL U3)(RETURN NIL))                                              00510
    ((MEMBER U3 (QUOTE(DONE NEXT)))(RETURN U3)))                            00520
  (RETURN N)                                                                00530
E2  (SETQ U6 (VERIFY FU))                                                   00540
  (PRINT FU)                                                                00550
  (PRINT(LIST(CADR FU)(QUOTE EQUALS) CX1))                                  00560
  (COND((EQUAL U6 (QUOTE DONE))(RETURN U6)))                                00570
E  (SETQ W (MASK Z))                                                        00580
  (SETQ QF2 NIL)                                                            00590
  (RETURN (QUOTE NEXT))                                                     00600
E1  (SETQ W (MASK Z))                                                       00610
  (SETQ QF2 NIL)                                                            00620
  (RETURN NIL) )))))                                                        00630
COMPILE((SOLVEX))                                                           00640
STOP)))                                                                     00650
```

```
                                SCNX        LISP
DEFINE(((SCNX(LAMBDA(S L U1 U U5)(PROG(U2 U3 U4)                          00010
  (SETQ U (GENSUBST(CDR U1)(CAR U1) U))                                   00020
  (SETQ U2 (LIST(QUOTE(EQUAL A1 A2))(QUOTE(EQUAL2 A1 A2))))               00030
A  (SETQ U3 (CAR U2))                                                     00040
  (SETQ U4 (RPLCE(QUOTE(A1 A2)) NIL))                                     00050
  (COND((NULL U4)(GO B)))                                                 00060
  (SETQ U3 (GENSUBST(CAR U4)(CDR U4) U3))                                 00070
B  (SETQ U4 (MATCH1 U3 U))                                                00080
B1  (COND((NULL U4)(GO C))                                                00090
    ((MEMBER L (CDAR U4))(GO D)))                                         00100
  (SETQ U4 (CDR U4))                                                      00110
  (GO B1)                                                                 00120
C  (SETQ U2 (CDR U2))                                                     00130
  (COND((NULL U2)(RETURN S)))                                             00140
  (GO A)                                                                  00150
D  (SETQ U3 (CDAR U4))                                                    00160
D1  (COND((EQUAL(CAR U3) N)(GO D2))                                       00170
    ((EQUAL(CAR U3) L)(GO D2)))                                           00180
  (GO E)                                                                  00190
D2  (SETQ U3 (CDR U3))                                                    00200
  (GO D1)                                                                 00210
E  (SETQ U4 (ALLVBLES(VBLES(CAR U3))))                                    00220
E1  (COND((NULL U4)(RETURN S))                                            00230
    ((FREEVBLE(CAR U4))(GO E2)))                                          00240
  (SETQ U4 (CDR U4))                                                      00250
  (GO E1)                                                                 00260
E2  (SETQ U2 (CAR U4))                                                    00270
  (SETQ U4 (CDR U5))                                                      00280
  (SETQ U4 (CONS(CAR U4)(CONS U2                                          00290
    (SUBST(CAR U3) L (CDDR U4)))))                                        00300
  (SETQ CX N)                                                             00310
  (SETQ L Z4)                                                             00320
  (SETQ U4 (PUTN2 NIL U4 (CADDDAR U5)))                                   00330
  (SETQ CX NIL)                                                           00340
  (COND((NULL U4)(RETURN S))                                              00350
    ((EQUAL L Z4)(RETURN S)))                                             00360
  (SETQ U1 (QUOTE(MEMBER A1 A)))                                          00370
  (SETQ U4 (RPLCE(QUOTE (A1 A)) NIL))                                     00380
  (COND((NULL U4)(GO G)))                                                 00390
  (SETQ U1 (GENSUBST(CAR U4)(CDR U4) U1))                                 00400
G  (SETQ U4 (MATCH1 U1 U))                                                00410
G1  (COND((NULL U4)(GO H))                                                00420
    ((MEMBER U2 (CDAR U4))(GO G2)))                                       00430
  (SETQ U4 (CDR U4))                                                      00440
  (GO G1)                                                                 00450
G2  (SETQ U1 (GENSUBST(CDAR U4)(CAAR U4) U1))                             00460
  (SETQ U4 (GENSYM))                                                      00470
  (SETQ Z3 (CONS(CONS N U4) Z3))                                          00480
  (SETQ U1 (CONS N (SUBST U4 U2 U1)))                                     00490
  (SETQ S (CDR(GFIND L)))                                                 00500
  (SETQ W (SUBST(SUBST U4 U2 S) S W))                                     00510
  (SETQ U3 (SUBST U4 U2 (CAR U3)))                                        00520
  (COND((NULL QF2)(SETQ QF2 (CONS U4 U3)))                                00530
    (T(SETQ QF2 (CONS U4 (SUBST U3 (CAR QF2)(CDR QF2))))))                00540
  (GO H1)                                                                 00550
H  (COND((NULL QF2)(SETQ QF2 (CONS U2 (CAR U3))))                         00560
    (T(SETQ QF2 (CONS U2 (SUBST(CAR U3)(CAR QF2)(CDR QF2))))))            00570
  (SETQ U1 N)                                                             00580
H1  (SETQ L (ROFW L))                                                     00590
H1A  (COND((NULL L)(RETURN U1)))                                          00600
  (SETQ S (SLVX(CDAR L)))                                                 00610
  (COND((EQ S N)(RETURN (QUOTE DONE))))                                   00620
  (SETQ L (CDR L))                                                        00630
  (GO H1A)                                                                00640
)))))       STOP))))                                                    00650
```

```
                          SLVX        LISP
DEFINE(((SLVX(LAMBDA(U)(PROG(U1 U2)                                          00010
  (COND((NULL QF2)(GO A1)))                                                  00020
  (SETQ CX1 (SOLVXP(CDADDR U)(CAR QF2)))                                     00030
  (COND((NULL CX1)(GO A1)))                                                  00040
  (SETQ CX1 (SUBST CX1 (CAR QF2)(CDR QF2)))                                  00050
  (RETURN N)                                                                 00060
A1  (SETQ CX1 (SOLVXP(CDADDR U)(CADR U)))                                    00070
  (COND((NOT(NULL CX1))(RETURN N)))                                          00080
  (SETQ U1 (CADDR U))                                                        00090
  (SETQ U2 (GMATCH(CADR U1)(CADDR U1)))                                      00100
  (COND((NULL U2)(RETURN NIL))                                               00110
    ((NOT(EQUAL(CAAR U2)(CADR U)))(RETURN NIL))                              00120
    ((NULL QF2)(SETQ CX1 (CADR U2)))                                         00130
    (T (SETQ CX1 (SUBST(CADR U2)(CAR QF2)(CDR QF2)))))                       00140
  (RETURN N) )))))                                                           00150
DEFINE(((SOLVXP(LAMBDA(U3 U)(PROG(U1 P PP)                                   00160
  (SETQ P (QUOTE *PROD))                                                     00170
  (SETQ PP (QUOTE *INVERSE))                                                 00180
A  (SETQ U1 (CAR U3))                                                        00190
  (COND((ATOM U1)(GO A1))                                                    00200
    ((EQUAL(CAR U1) P)(GO B1))                                               00210
    ((EQUAL(CAR U1) PP)(GO C1)))                                             00220
  (RETURN NIL)                                                               00230
A1  (COND((EQUAL U1 U)(RETURN(CADR U3))))                                    00240
  (RETURN NIL)                                                               00250
B1  (COND((MEMBER U (VBLES(CADR U1)))(GO B2)))                               00260
  (SETQ U3 (LIST(CADDR U1)(LIST P (LIST PP (CADR U1)                         00270
    (CADDDR U1))(CADR U3)(CADDDR U1))))                                      00280
  (GO A)                                                                     00290
B2  (COND((MEMBER U (VBLES(CADDR U1)))(RETURN NIL)))                         00300
  (SETQ U3 (LIST(CADR U1)(LIST P (CADR U3)(LIST PP (CADDR U1)                00310
    (CADDDR U1))(CADDDR U1))))                                               00320
  (GO A)                                                                     00330
C1  (SETQ U3 (LIST(CADR U1)(LIST PP (CADR U3)(CADDR U1))))                   00340
  (GO A) )))))                                                               00350
COMPILE((SOLVXP))      STOP))))))                                            00360
```

```
                          HOMOMF        LISP
DEFINE(((HOMOMF(LAMBDA(L)(PROG(U3 U4)                                        00010
  (SETQ U3 Z4)                                                               00020
  (SETQ U4 (PUTN2 NIL (CADR L) NIL))                                         00030
  (SETQ U4 (SCANW NIL(QUOTE GEN)))                                           00040
  (SETQ QF1 (ONLY(ROFW U3)(QUOTE(REL REL1))))                                00050
  (SETQ W (MASK U3))                                                         00060
  (SETQ U3 Z4)                                                               00070
  (SETQ U4 (PUTN2 NIL (CADDR L) NIL))                                        00080
  (COND((NOT(EQUAL U4 (QUOTE DONE)))(SETQ U4 (SCANW NIL (QUOTE GEN)))))      00090
  (SETQ QF1 NIL)                                                             00100
  (SETQ W (MASK U3))                                                         00110
  (COND((EQUAL U4 (QUOTE DONE))(SETQ U4 (VERIFY L))))                        00120
  (RETURN U4) )))))                                                          00130
DEFINE(((ONLY(LAMBDA(U L)                                                    00140
(COND((NULL U) NIL)((MEMBER(CAAAR U) L)(CONS(CAR U)                          00150
(ONLY(CDR U) L)))(T (ONLY(CDR U) L))) ))))                                   00160
STOP))))                                                                     00170
```

```
                              GENFCN        LISP
DEFINE(((GENFCN(LAMBDA(L)(PROG(U U1 U2 U3 U4 U5 U6 S)                    00010
  (SETQ U4 (CONS(GENSYM)(GENSYM)))                                       00020
  (SETQ Z3 (CONS(CONS N (CAR U4)) Z3))                                   00030
  (SETQ Z2 (CONS(CONS(QUOTE RES)(CDR U4)) Z2))                           0004C
  (COND((EQUAL(CAR L)(CADR L))(GO E)))                                   00050
  (SETQ U6 (CADR L))                                                     00060
A1  (SETQ U3 Z4)                                                         00070
  (SETQ U (GENSYM))                                                      00080
(SETQ Z3 (CONS(CONS(QUOTE GEN) U) Z3))                                   00090
  (SETQ U1 (PUTQN1(CONS Z4A Z4B)(LIST(QUOTE MEMBER) U U6)))              00100
  (SETQ U1 (PUTN2 NIL (LIST(QUOTE DUMMY) U) NIL))                        00110
  (SETQ U1 (SCANW N (QUOTE GEN)))                                        00120
  (SETQ U1 (CDR(ROFW U3)))                                               00130
  (SETQ W (MASK U3))                                                     00140
  (SETQ U3 (MEMOF1 VV1 (QUOTE MEMBER)))                                  00150
  (COND((NULL S)(SETQ U2 (CONS(LIST U6 U U) NIL))))                      00160
A2  (COND((NULL U1)(GO D2)))                                             00170
  (SETQ U (GENVBLE(CDAR U1)))                                            00180
  (COND((NULL U)(GO D3)))                                                00190
  (SETQ U5 U3)                                                           00200
A5  (COND((NULL U5)(GO D3))                                              00210
    ((EQUAL U (CAAR U5))(GO A6)))                                        00220
  (SETQ U5 (CDR U5))                                                     00230
  (GO A5)                                                                00240
A6  (COND((NULL S)(GO S2)))                                              00250
  (SETQ U5 (CADAR U5))                                                   00260
  (SETQ U6 U2)                                                           00270
A7  (COND((NULL U6)(GO A8))                                              00280
    ((EQUAL U5 (CAAR U6))(GO D)))                                        00290
  (SETQ U6 (CDR U6))                                                     00300
  (GO A7)                                                                00310
S2  (COND((EQUAL(CAR L)(CADAR U5))(GO C)))                               00320
  (SETQ U2 (APPEND U2 (CONS(LIST(CADAR U5) U                             00330
    (CADDAR U1)) NIL)))                                                  00340
  (GO A8)                                                                00350
D3  (COND((NULL S)(SETQ U2 (APPEND U2 (LIST(LIST N))))))                 00360
A8  (SETQ U1 (CDR U1))                                                   00370
  (GO A2)                                                                00380
D2  (COND((EQ S N)(RETURN NIL)))                                         00390
  (SETQ S N)                                                             00400
  (SETQ U6 (CAR L))                                                      00410
  (GO A1)                                                                00420
C   (SETQ U6 (SUBST(CAR U4) U (CADDAR U1)))                              00430
C1  (RETURN(LIST(QUOTE IMPLIES)(LIST(QUOTE MEMBER)                       00440
    (CAR U4)(CAR L))(LIST(QUOTE EQUAL)(LIST(CDR U4)                      00450
    (CAR L)(CADR L))(CAR U4)) U6)))                                      00460
D   (RETURN(LIST(QUOTE EQUAL)(LIST(CDR U4)(CAR L)                        00470
    (CADR L)(SUBST(CAR U4) U (CADDAR U1)))                               00480
    (SUBST(CAR U4)(CADAR U6)(CADDAR U6))))                               00490
E   (SETQ U6 (CAR U4))                                                   00500
  (GO C1)                                                                00510
)))))     STOP))))))                                                    00520
```

```
                          WLDFN        LISP
DEFINE(((COMPOSITION(LAMBDA(U VB)(PROG(U1)                                    00010
A  (COND((ATOM U)(RETURN NIL))                                                00020
    ((ON(CAR U) Z2)(SETQ U1 (CONS(LIST(CAR U)(CADR U)                         00030
      (CADDR U)) U1)))                                                        00040
    (T (RETURN NIL)))                                                         00050
  (SETQ U (CADDDR U))                                                         00060
  (COND((EQUAL U VB)(RETURN U1)))                                             00070
  (GO A) )))))                                                                00080
DEFINE(((WLDFN(LAMBDA(EQN)(PROG(U1 U2)                                        00090
  (PRINT(QUOTE(MUST PROVE RELATION WELL DEFINED)))                            00100
  (PRINT(QUOTE(IS INVERSE RELATION ONE TO ONE)))                              00110
  (SETQ U1 (GENSYM))                                                          00120
  (SETQ U2 (CDAR EQN))                                                        00130
  (SETQ Z2 (CONS(CONS(QUOTE RES) U1) Z2))                                     00140
  (SETQ U1 (MONOMF(LIST(QUOTE EQUAL)(LIST U1 (CADR U2)                        00150
    (CAR U2)(CADR EQN))(CADDR U2)) NIL))                                      00160
  (COND((NULL U1)(RETURN NIL)))                                               00170
  (SETQ Z6 NIL)                                                               00180
  (SETQ LON NIL)                                                              00190
  (SETQ W1 NIL)                                                               00200
  (SETQ W NIL)                                                                00210
  (RETURN T) )))))                                                            00220
DEFINE(((WLDFN1(LAMBDA(U4 U5 U6 U7)(PROG(FV S U1 U2 U3)                       00230
  (SETQ U1 (CONS(GENSYM)(GENSYM)))                                            00240
  (SETQ Z3 (CONS(CONS(QUOTE RES)(CAR U1))(CONS(CONS(QUOTE RES)                00250
    (CDR U1)) Z3)))                                                           00260
  (SETQ U3 (GENVBLE U4))                                                      00270
C0  (COND((ATOM U5)(GO C1))                                                   00280
    ((EQUAL(CAR U5)(QUOTE *LCOSET))(GO C4)))                                  00290
C1  (SETQ U2 (LIST(QUOTE EQUAL)(LIST(QUOTE *PROD)(LIST(QUOTE *INVERSE)        00300
    (SUBST(CAR U1) U3 U5)(CADR U6))(SUBST(CDR U1) U3 U5)                      00310
    (CADR U6))(LIST(QUOTE *IDENTITY)(CADR U6))))                              00320
C2  (COND((NULL S)(GO C3))                                                    00330
    ((NULL QM5)(GO C2A)))                                                     00340
  (SETQ FV (LIST(QUOTE MEMBER) U3 (GENSET U3 U7)))                            00350
  (SETQ FV (PUTON1(CONS Z4A Z4B)(LIST(QUOTE AND)(SUBST(CAR U1) U3 FV)         00360
    (SUBST(CDR U1) U3 FV))))                                                  00370
  (SETQ Z4B (ADD1 Z4B))                                                       00380
C2A  (SETQ FV (SOLVE(LIST(QUOTE IMPLIES)U2 S)))                               00390
  (COND((NULL FV)(RETURN NIL)))                                               00400
  (RETURN N)                                                                  00410
C3  (SETQ S U2)                                                               00420
  (SETQ U5 U4)                                                                00430
  (SETQ U6 (CDR U6))                                                          00440
  (GO C0)                                                                     00450
C4  (SETQ U2 (LIST(QUOTE MEMBER)(LIST(QUOTE *PROD)(LIST                       00460
    (QUOTE *INVERSE)(SUBST(CAR U1) U3 (CADR U5))(CADDDR U5))                  00470
    (SUBST(CDR U1) U3 (CADR U5))(CADDDR U5))(CADDR U5)))                      00480
  (GO C2)                                                                     00490
)))))    STOP)))))))                                                          00500
```

```
                          MONOMF        LISP
DEFINE(((MONOMF(LAMBDA(EQN S)(PROG(U1 U2 U3 U4 U5 U6 U7 FV)                  00010
  (SETQ U6 (CADR EQN))                                                       00020
  (COND((FREEVBLE(CADDDR U6))(GO B)))                                        00030
A1  (SETQ U2 (CONS Z4A Z4))                                                  00040
  (SETQ FV (CONS VV1 VV2))                                                   00050
  (SETQ U4 (GENSYM))                                                         00060
  (SETQ Z3 (CONS(CONS(QUOTE GEN) U4) Z3))                                    00070
  (SETQ U3 (PUTON1(CONS Z4A Z4B)(LIST(QUOTE MEMBER) U4 (CADR U6))))          00080
  (SETQ U3 (PUTN2 NIL (LIST(QUOTE DUMMY)(SUBST U4                            00090
    (CADDDR U6) U6)) NIL))                                                   00100
  (SETQ U3 (SCANW N (QUOTE GEN)))                                            00110
  (SETQ U3 Z4)                                                               00120
  (SETQ VV1 NIL)(SETQ VV2 NIL)                                               00130
  (SETQ U4 (PUTON1 (CONS Z4A Z4B) EQN))                                      00140
  (SETQ U4 (SCANW NIL (QUOTE GEN)))                                          00150
  (SETQ U4 (GFIND U3))                                                       00160
  (COND((NULL U4)(RETURN NIL)))                                              00170
  (SETQ U5 (CADDR(CDADDR(GFIND(CAADAR U4)))))                                00180
  (SETQ U4 (CADDR U4))                                                       00190
  (SETQ Z6 NIL)                                                              00200
  (SETQ LON NIL)                                                             00210
  (SETQ W1 NIL)                                                              00220
  (SETQ W NIL)                                                               00230
  (SETQ Z4 (CDR U2))(SETQ Z4A (CAR U2))                                      00240
  (SETQ VV1 (CAR FV))(SETQ VV2 (CDR FV))                                     00250
  (SETQ U7 QM2)                                                              00260
  (SETQ QM2 FQM2)(SETQ QM3 FQM3)(SETQ QM4 FQM4)(SETQ QM6 NIL)                00270
  (COND((NULL S)(RETURN(WLDFN1 U4 U5 U6 U7)))                                00280
    ((ATOM U4)(GO A2))                                                       00290
    ((EQUAL(CAR U4)(QUOTE *LCOSET))(GO A6)))                                 00300
A2  (SETQ U1 (GENVBLE U4))                                                   00310
  (COND((NULL U1)(GO A8)))                                                   00320
  (SETQ U3 (SOLVXP(LIST U4 (LIST(QUOTE *IDENTITY)                            00330
    (CADDR U6))) U1))                                                        00340
  (COND((NULL U3)(GO A8)))                                                   00350
  (SETQ FV (PUTON1(CONS Z4A Z4B)(LIST(QUOTE EQUAL) U1 U3)))                  00360
  (GO D)                                                                     00370
A6  (SETQ FV (PUTON1(CONS Z4A Z4B)(LIST(QUOTE MEMBER)                        00380
    (CADR U4)(CADDR U4))))                                                   00390
  (SETQ U1 (GENVBLE U4))                                                     00400
  (GO D)                                                                     00410
A8  (SETQ FV (PUTON1(CONS Z4A Z4B)(LIST(QUOTE EQUAL) U4                      00420
    (LIST(QUOTE *IDENTITY)(CADDR U6)))))                                     00430
D  (SETQ Z4B (ADD1 Z4B))                                                     00440
  (COND((NULL QM5)(GO D0))                                                   00450
    ((NULL U1)(GO D0)))                                                      00460
  (SETQ FV (PUTON1(CONS Z4A Z4B)(LIST(QUOTE MEMBER) U1 (GENSET U1 U7))))     00470
  (SETQ Z4B (ADD1 Z4B))                                                      00480
D0  (SETQ FV (SOLVE(LIST(QUOTE EQUAL) U5                                     00490
    (LIST(QUOTE *IDENTITY)(CADR U6)))))                                      00500
  (COND((NULL FV)(RETURN NIL)))                                              00510
  (RETURN N)                                                                 00520
B  (SETQ U1 (COMPOSITION(CADDR EQN)(CADDDR U6)))                             00530
  (COND((NULL U1)(GO A1)))                                                   00540
B1  (COND((ON(LIST(QUOTE ONEONE)(CAR U1)) VV1)(SETQ U1 (CDR U1)))            00550
    (T (GO A1)))                                                             00560
  (COND((NULL U1)(GO B2)))                                                   00570
  (GO B1)                                                                    00580
B2  (PRINT(QUOTE(FUNCTION IS A COMPOSITION OF MONOMORPHISMS)))               00590
  (RETURN N)                                                                 00600
)))))    STOP)))))))                                                       00610
```

```
                        VERIFY        LISP
DEFINE(((VERIFY(LAMBDA(U)                                                  00010
(PROG(U1 U2 U3 U4 U5 U6 U7 TT NN G)                                        00020
  (SETQ G (QUOTE A))                                                       00030
A0  (SETQ U2 W1)                                                           00040
A1  (COND((NULL U2)(GO A2))                                                00050
    ((EQUAL G (QUOTE A))(GO A1A))                                          00060
    ((EQUAL(CAR U6)(CADDDAAR U2))(GO A3)))                                 00070
A1B  (SETQ U1 (APPEND U1 (CONS(CAR U2) NIL)))                              00080
  (SETQ U2 (CDR U2))                                                       00090
  (GO A1)                                                                  00100
A1A  (COND((EQUAL U (CDAR U2))(GO A3)))                                    00110
  (GO A1B)                                                                 00120
A3  (SETQ U3 (CAAAR U2))                                                   00130
  (COND((MEMBER U3 (QUOTE(VER VERA2 IRR)))(GO A6))                         00140
    ((EQUAL G (QUOTE A))(GO A3C))                                          00150
    ((EQUAL G (QUOTE B))(GO A3D))                                          00160
    ((ONLYMEMB(CDR U6)(CADAAR U2))(GO A3C1)))                              00170
A3C3  (SETQ U7 (DELTF(CDR U6)(CADAAR U2)))                                 00180
  (SETQ U3 (CONS U3 (CONS U7 (CDDAAR U2))))                                00190
A3E  (SETQ U2 (CONS(CONS U3 (CDAR U2))(CDR U2)))                           00200
A9B  (COND((NULL TT)(SETQ W1 (APPEND U1 U2)))                              00210
  (T (SETQ W (APPEND U1 U2))))                                             00220
  (GO A7)                                                                  00230
A3D  (COND((MEMBER(CDR U6)(CADDAAR U2))(GO A3C)))                          00240
  (SETQ U7 (DELPR(CDR U6)(CADDAAR U2)))                                    00250
  (SETQ U3 (CONS U3 (CONS(CADAAR U2)(CONS U7(CDR(CDDAAR U2))))))           00260
  (GO A3E)                                                                 00270
A3C2  (COND((MEMBER(QUOTE HEAD)(CADAAR U2))(GO A3C3)))                     00280
  (GO A3C)                                                                 00290
A3C1  (COND((EQUAL U3 (QUOTE A2))(GO A3C2)))                               00300
A3C  (SETQ U3 (CAR U2))                                                    00310
  (SETQ U2 (CDR U2))                                                       00320
  (COND((EQUAL G (QUOTE C))(GO A3F))                                       00330
    ((EQUAL G (QUOTE A))(SETQ U6 (CONS(CADDDAR U3) NIL))))                 00340
  (SETQ U4 (MIRGEA(TACK(CADAR U3)(CAR U6)) U4))                            00350
  (SETQ U5 (MIRGEA(TACK(CADDAR U3)(CAR U6)) U5))                           00360
  (COND((EQUAL(CAAR U3)(QUOTE A2))(GO A4))                                 00370
    ((EQUAL(CAAR U3)(QUOTE PNT))(SETQ NN (QUOTE NEXT))))                   00380
  (SETQ U3 (CONS(CONS(QUOTE VER)(CDAR U3))(CDR U3)))                       00390
A3A  (SETQ U2 (CONS U3 U2))                                                00400
  (GO A9B)                                                                 00410
A3F  (SETQ U5 (MIRGEA(TACK(CADDAR U3)(CAR U6)) U5))                        00420
  (COND((EQUAL(CAAR U3)(QUOTE PNT))(SETQ NN (QUOTE NEXT))))                00430
  (SETQ U3 (CONS(CONS(QUOTE IRR)(CDAR U3))(CONS N (CDR U3))))              00440
  (GO A3A)                                                                 00450
A4  (SETQ U3 (CONS(CONS(QUOTE VERA2)(CDAR U3))(CDR U3)))                   00460
  (GO A3A)                                                                 00470
A2  (COND((NULL TT)(GO A2A)))                                              00480
  (GO A7)                                                                  00490
A2A  (SETQ TT N)                                                           00500
  (SETQ U1 NIL)                                                            00510
  (SETQ U2 W)                                                              00520
  (GO A1)                                                                  00530
A6  (COND((EQUAL G (QUOTE A))(RETURN NIL)))                                00540
A7  (COND((EQUAL G (QUOTE C))(GO A7A)))                                    00550
  (SETQ G (QUOTE B))                                                       00560
  (COND((NULL U4)(GO A7A)))                                                00570
  (SETQ U6 (CAR U4))                                                       00580
  (SETQ U4 (CDR U4))                                                       00590
A7B  (SETQ TT NIL)                                                         00600
  (SETQ U1 NIL)                                                            00610
  (GO A0)                                                                  00620
A7A  (COND((NULL U5)(GO C1)))                                              00630
  (SETQ U6 (CAR U5))                                                       00640
  (SETQ U5 (CDR U5))                                                       00650
  (SETQ G (QUOTE C))                                                       00660
  (GO A7B)                                                                 00670
C1  (COND((NULL W1)(GO C2))                                                00680
    ((EQUAL(CAAAR W1)(QUOTE VER))(RETURN(QUOTE DONE))))                    00690
  (RETURN NN)                                                              00700
C2  (COND((EQUAL(CAAAR W)(QUOTE VER))(RETURN(QUOTE DONE))))                00710
  (RETURN NN) )))))                                                        00720
COMPILE((VERIFY))                                                          00730
STOP)                                                                      00740
```

```
                              INVM        LISP
DEFINE(((INVM(LAMBDA(U U2)(PROG(U1 U4 U5 LL)                                  00010
  (SETQ LL Q)                                                                 00020
  (COND((MEMBER(CAR U)(QUOTE(IMPLIES FEQUAL EXISTS)))                         00030
     (RETURN NIL))                                                            00040
    ((NOT(AND(MEMBER(QUOTE *PROD) U2)(MEMBER(QUOTE *INVERSE) U2)))            00050
     (RETURN(INVM2 U U5 U2))))                                                00060
B4  (COND((NULL LL)(RETURN(INVM2 U U5 U2))))                                  00070
  (SETQ U1 (PMATCH(CAR LL) U))                                                00080
B3  (COND((NULL U1)(GO B1)))                                                  00090
  (SETQ U4 (INVSUB(PRODARGS(CDAR U1))))                                       00100
  (COND((NULL U4)(GO B2)))                                                    00110
  (SETQ U (INVREDO(CDAR U1) U4 U (CAR LL)))                                   00120
  (SETQ U5 U)                                                                 00130
B2  (SETQ U1 (CDR U1))                                                        00140
  (GO B3)                                                                     00150
B1  (SETQ LL (CDR LL))                                                        00160
  (GO B4) )))))                                                               00170
DEFINE(((INVM2(LAMBDA(U U5 U6)(PROG(U1 U2 U3 U4 S1)                           00180
  (COND((NULL U5)(SETQ S1 N)))                                                00190
(COND((MEMBER(QUOTE *INVERSE) U6)(GO E1)))                                    00200
  (SETQ U1 (LIST NIL NIL))                                                    00210
  (SETQ U2 (LIST NIL))                                                        00220
  (SETQ U3 (LIST NIL))                                                        00230
  (COND((AND(MEMBER(QUOTE *PROD) U6)(MEMBER(QUOTE *IDENTITY) U6))             00240
    (SETQ U4 (LIST N N)))                                                     00250
    (T (SETQ U4 (LIST NIL NIL))))                                             00260
  (GO E9)                                                                     00270
E1  (SETQ U2 (LIST N))                                                        00280
  (COND((MEMBER(QUOTE *PROD) U6)(GO E2)))                                     00290
  (SETQ U1 (LIST NIL NIL))                                                    00300
  (SETQ U4 (LIST NIL NIL))                                                    00310
  (COND((MEMBER(QUOTE *IDENTITY) U6)(SETQ U3 (LIST N)))                       00320
    (T (SETQ U3 (LIST NIL))))                                                 00330
  (GO E9)                                                                     00340
E2  (SETQ U1 (LIST N N))                                                      00350
  (COND((MEMBER(QUOTE *IDENTITY) U6)(GO E3)))                                 00360
  (SETQ U3 (LIST NIL))                                                        00370
  (SETQ U4 (LIST NIL NIL))                                                    00380
  (GO E9)                                                                     00390
E3  (SETQ U3 (LIST N))                                                        00400
  (SETQ U4 (LIST N N))                                                        00410
E9  (SETQ U3 (SIFT(APPEND U1 (APPEND U2 (APPEND U3 U4)))))                    00420
A  (COND((NULL U3)(GO D)))                                                    00430
  (SETQ U2 (CAAR U3))                                                         00440
  (SETQ U5 (CADAR U3))                                                        00450
  (SETQ U1 (RPLCE(QUOTE(A1 A)) NIL))                                          00460
  (COND((NULL U1)(GO B)))                                                     00470
  (SETQ U2 (GENSUBST(CAR U1)(CDR U1) U2))                                     00480
  (SETQ U5 (GENSUBST(CAR U1)(CDR U1) U5))                                     00490
B  (SETQ U1 (MATCH1 U2 U))                                                    00500
  (COND((NULL U1)(GO C)))                                                     00510
  (SETQ S1 NIL)                                                               00520
  (SETQ U (RPTSUB U5 U2 U U1))                                                00530
C  (SETQ U3 (CDR U3))                                                        00540
  (GO A)                                                                      00550
D  (COND((NULL S1)(RETURN U)))                                                00560
  (RETURN NIL) )))))                                                          00570
DEFINE(((SIFT(LAMBDA(U)(PROG(U1 U2)                                           00580
  (SETQ U1 (QUOTE(((*PROD A1 (*INVERSE A1 A) A)                               00590
    (*IDENTITY A))((*PROD(*INVERSE A1 A) A1 A)                                00600
    (*IDENTITY A))((*INVERSE(*INVERSE A1 A) A) A1)                            00610
    ((*INVERSE(*IDENTITY A) A)(*IDENTITY A))                                  00620
    ((*PROD A1 (*IDENTITY A) A) A1)                                           00630
    ((*PROD (*IDENTITY A) A1 A) A1))))                                        00640
A  (COND((NULL U)(RETURN U2))                                                 00650
    ((EQUAL(CAR U) N)(SETQ U2 (CONS(CAR U1)U2))))                             00660
  (SETQ U (CDR U))                                                            00670
  (SETQ U1 (CDR U1))                                                          00680
  (GO A) )))))                                                                00690
COMPILE((INVM))                                                               00700
STOP))))                                                                      00710
```

```
                          MODEL        LISP
DEFINE(((MODELMAKE(LAMBDA(U)(PROG(U1 U2 U3)                                    00010
  (SETQ U1 QM1)                                                                00020
A0  (COND((NULL U1)(RETURN NIL))                                               00030
    ((EQUAL(CAR U)(CAAAR U1))(GO A1)))                                         00040
  (SETQ U1 (CDR U1))                                                           00050
  (GO A0)                                                                      00060
A1  (SETQ U1 (CAR U1))                                                         00070
  (SETQ U2 (RPLCE(ALLVBLES(VBLES(CDAR U1)))(ALLVBLES(VBLES(CDR U)))))          00080
  (COND((NULL U2)(GO A1B)))                                                    00090
  (SETQ U1 (GENSUBST(CAR U2)(CDR U2) U1))                                      00100
A1B  (SETQ U1 (GENSUBST(CDR U)(CDAR U1)(CDR U1)))                              00110
  (SETQ U2 (CAR U1))                                                           00120
A2  (COND((NULL U2)(GO A3)))                                                   00130
  (SETQ U3 (ALLVBLES(CAR U2)))                                                 00140
  (SETQ U QM3)                                                                 00150
  (SETQ QM3 NIL)                                                               00160
A2A  (COND((NULL U)(GO A2C))                                                   00170
    ((NULL(MEET(CAR U) U3))(SETQ QM3 (CONS(CAR U) QM3)))                       00180
    (T (GO A2B)))                                                              00190
A2A1  (SETQ U (CDR U))                                                         00200
  (GO A2A)                                                                     00210
A2B  (SETQ U3 (ALLVBLES(APPEND(CAR U) U3)))                                    00220
  (GO A2A1)                                                                    00230
A2C  (SETQ QM3 (CONS U3 QM3))                                                  00240
  (SETQ U2 (CDR U2))                                                           00250
  (GO A2)                                                                      00260
A3  (SETQ U1 (CDR U1))                                                         00270
A3A  (COND((NULL U1)(RETURN NIL))                                              00280
    ((EQUAL(CAAR U1)(CADAR U1))(GO A3B)))                                      00290
  (SETQ QM4 (SWMEMB(CAR U1)(CONS(CADAR U1)(CONS(CAAR U1) NIL)) QM4))           00300
A3B  (SETQ U1 (CDR U1))                                                        00310
  (GO A3A) )))))                                                               00320
DEFINE(((CHECKMODEL(LAMBDA(U)(PROG(U1 U2 U3)                                   00330
  (SETQ U1 QM2)                                                                00340
A0  (COND((NULL U1)(GO A1))                                                    00350
    ((EQUAL(CAAR U1)(CAR U))(SETQ U2 (CONS(CADAR U1) U2))))                    00360
  (SETQ U1 (CDR U1))                                                           00370
  (GO A0)                                                                      00380
A1  (COND((NULL U2)(RETURN T)))                                                00390
  (SETQ U1 QM3)                                                                00400
A1A  (COND((NULL U1)(RETURN T))                                                00410
    ((MEMBER(CADR U)(CAR U1))(GO A2)))                                         00420
  (SETQ U1 (CDR U1))                                                           00430
  (GO A1A)                                                                     00440
A2  (SETQ U1 (CAR U1))                                                         00450
  (SETQ U3 U2)                                                                 00460
A2A  (COND((NULL U2)(RETURN NIL))                                              00470
    ((MEMBER(CAR U2) U1)(GO A3)))                                              00480
  (SETQ U2 (CDR U2))                                                           00490
  (GO A2A)                                                                     00500
A3  (SETQ U1 QM4)                                                              00510
  (SETQ U2 NIL)                                                                00520
A3A  (COND((NULL U1)(GO A4))                                                   00530
    ((EQUAL(CAAR U1)(CADR U))(SETQ U2 (CONS(CADAR U1) U2))))                   00540
  (SETQ U1 (CDR U1))                                                           00550
  (GO A3A)                                                                     00560
A4  (COND((NULL U2)(RETURN T)))                                                00570
A4A  (COND((NULL U3)(RETURN NIL))                                              00580
    ((MEMBER(CAR U3) U2)(SETQ U3 (CDR U3)))                                    00590
    (T (RETURN T)))                                                            00600
  (GO A4A) )))))                                                               00610
DEFINE(((MEET(LAMBDA(U1 U2)                                                    00620
  (COND((NULL U1) NIL)((MEMBER(CAR U1) U2)(CONS(CAR U1)(MEET                   00630
  (CDR U1) U2)))(T (MEET(CDR U1) U2))) ))))                                    00640
DEFINE(((SWMEMB(LAMBDA(U1 U2 L)                                                00650
  (COND((NULL L)(CONS U1 NIL))((EQUAL U1 (CAR L)) L)                           00660
  ((EQUAL U2 (CAR L))(CDR L))(T (CONS(CAR L)(SWMEMB U1 U2(CDR L))))))))))      00670
DEFINE(((NOWWHICH(LAMBDA(U)(PROG(U1)                                           00680
  (SETQ U1 (CDADR U))                                                          00690
  (COND((MEMBER U1 QM6)(RETURN U))                                             00700
    ((MEMBER U1 QM2)(GO A))                                                    00710
    ((NULL (CHECKMODEL U1))(RETURN U)))                                        00720
A  (RETURN(LIST(CAR U)(CADDR U)(CADR U))) )))))                                00730
DEFINE(((GENSET(LAMBDA(U L)(PROG()                                             00740
A  (COND((NULL L)(GO B))                                                       00750
    ((EQUAL(CAAR L) U)(RETURN(CADAR L))))                                      00760
  (SETQ L (CDR L))                                                             00770
  (GO A)                                                                       00780
B  (PRINT(QUOTE(GENSET IS RETURNING NONSENSE)))                                00790
  (RETURN(QUOTE XX)) )))))                                                     00800
STOP))))                                                                       00810
```

```
                          GTFRM        LISP
DEFINE(((GTFRM(LAMBDA(U L Z LN)                                          00010
(PROG(U1 U2 U3 U4 S)                                                     00020
A  (COND((EQUAL L (QUOTE Y))(SETQ U4 Y))                                 00030
    ((EQUAL L (QUOTE X))(SETQ U4 X))                                     00040
    (T (SETQ U4 XX)))                                                    00050
A1  (COND((NULL U4)(GO A2))                                              00060
    ((EQUAL(CAAAR U4)(CAR U))(GO A11)))                                  00070
  (SETQ U4 (CDR U4))                                                     00080
  (GO A1)                                                                00090
A11  (COND((NULL(CDAR U4))(GO A1B)))                                     00100
  (SETQ U1 (CDAAR U4))                                                   00110
  (SETQ U2 (CDAR U4))                                                    00120
  (SETQ U3 (ALLVBLES(VBLES U2)))                                         00130
  (SETQ U3 (RPLCEF U3 (ALLVBLES(VBLES(CDR U)))))                         00140
  (COND((NULL U3)(GO A1A)))                                              00150
  (SETQ U2 (GENSUBST(CAR U3)(CDR U3) U2))                                00160
  (SETQ U1 (GENSUBST(CAR U3)(CDR U3) U1))                                00170
A1A  (SETQ U2 (GENSUBST(CDR U) U1 U2))                                   00180
  (COND((OR(EQUAL L (QUOTE X))(EQUAL S N))(RETURN                        00190
      (PUTN2 NIL U2 Z)))                                                 00200
    ((EQUAL L (QUOTE XX))(GO B))                                         00210
    (T (RETURN(PUTON1(CONS Z4A LN) U2))))                                00220
A1B  (COND((EQUAL L (QUOTE X))(SETQ S N))                                00230
    (T (RETURN NIL)))                                                    00240
  (SETQ L (QUOTE Y))                                                     00250
  (GO A)                                                                 00260
A2  (COND((EQUAL L (QUOTE X))(PRINT(QUOTE                                00270
    (CAN YOU GIVE ME A SUFFICIENT CONDITION FOR))))                      00280
    (T (PRINT(QUOTE (CAN YOU GIVE ME A DEFINITION OF)))))                00290
  (PRINT(CAR U))                                                         00300
  (SETQ U1 (RDFLX()))                                                    00310
  (COND((EQUAL U1 (QUOTE NO))(SETQ U1 (CONS(CONS(CAR U)                  00320
     NIL) NIL))))                                                        00330
  (COND((EQUAL L(QUOTE Y))(SETQ Y (CONS U1 Y)))                          00340
    ((EQUAL L(QUOTE X))(SETQ X (CONS U1 X)))                             00350
    (T (SETQ XX(CONS U1 XX))))                                           00360
  (SETQ U4 (CONS U1 NIL))                                                00370
  (GO A11)                                                               00380
B  (COND((NOT(ON U2 LN))(SETQ LN (APPEND LN                              00390
     (CONS(CONS N U2) NIL)))))                                           00400
  (RETURN LN)                                                            00410
))))) STOP))))                                                           00420
```

```
DEFINE(((ADDCONSQ(LAMBDA(Z M)(PROG(U3 U4 U5)
  (SETQ U3 (CAR(GFIND M)))
  (SETQ U5 (APPENDM(CADDR U3) Z))
  (COND((NULL U5)(RETURN NIL)))
  (SETQ U4 (CONS(CAR U3)(CONS(CADR U3)(CONS U5 (CDDDR U3)))))
  (COND((OR(NULL W)(LESSP M (CADDDAAR W)))
      (SETQ W1 (SUBST U4 U3 W1)))
    (T (SETQ W (SUBST U4 U3 W))))
  (RETURN NIL) )))))
DEFINE(((APPENDM(LAMBDA(L Z)
  (COND((MEMBER Z L) NIL) (T (CONS Z L))) ))))
DEFINE(((ADDANTEC(LAMBDA(M U)(PROG(U3 U4 U5)
  (SETQ U3 (CAR(GFIND U)))
  (COND((GMEMB M (CADDR U3))(RETURN NIL))
    ((EQUAL M (CADDDR U3))(RETURN NIL))
    ((NULL(ANDEND M (CADDR U3)))(RETURN NIL))
    ((MEMBER(CAR U3)(QUOTE(VER VERA2)))(RETURN(QUOTE B))))
A2A  (SETQ U5 (APPENDM(CADR U3) M))
  (COND((NULL U5)(RETURN NIL)))
  (SETQ U4 (CONS(CAR U3)(CONS U5 (CDDR U3))))
  (COND((OR(NULL W)(LESSP(CADDDR U3)(CADDDAAR W)))
    (SETQ W1 (SUBST U4 U3 W1)))
    (T (SETQ W (SUBST U4 U3 W))))
  (RETURN (CADDDR U3)) )))))
DEFINE(((ANDARGS(LAMBDA(U)(PROG()
  (COND((EQUAL(CAADR U)(QUOTE AND))(GO A1))
    ((EQUAL(CAADDR U)(QUOTE AND))(GO A2)))
  (RETURN(CDR U))
A1  (COND((EQUAL(CAADDR U)(QUOTE AND))(GO A3)))
  (RETURN(APPEND(CDR U)(ANDARGS(CADR U))))
A2  (RETURN(APPEND(CDR U)(ANDARGS(CADDR U))))
A3  (RETURN(APPEND(CDR U)(APPEND(ANDARGS(CADR U))(ANDARGS(CADDR U))))) )
))))
DEFINE(((ON2AVER(LAMBDA(U)(PROG(U1)
  (SETQ U1 (GFIND U))
  (COND((NULL U1)(RETURN NIL))
    ((EQUAL(CAAR U1)(QUOTE VERA2))(RETURN(CADAR U1))))
  (RETURN NIL) )))))
DEFINE(((MATCH1(LAMBDA(U L)(PROG(U1 U2)
  (COND((NULL L)(RETURN NIL))
    ((ATOM(CAR L))(GO A2)))
  (SETQ U1 (MATCH1 U (CAR L)))
  (COND((NULL U1)(RETURN(MATCH1 U (CDR L)))))
  (SETQ U2 (MATCH1 U (CDR L)))
  (COND((NULL U2)(RETURN U1)))
  (RETURN (APPEND U1 U2))
A2  (COND((ATOM U)(GO A5))
  ((FREEVBLE(CAR U))(GO A1))
    ((EQUAL(CAR U)(CAR L))(GO A5)))
  (RETURN(MATCH1 U (CDR L)))
A1  (COND((ON(CAR L) Z2)(GO A5)))
  (RETURN(MATCH1 U (CDR L)))
A5  (SETQ U1 (GMATCH U L))
  (COND((NULL U1)(RETURN (MATCH1 U (CDR L)))))
  (SETQ U2 (MATCH1 U (CDR L)))
  (RETURN (CONS U1 U2)) )))))
COMPILE((ADDCONSQ APPENDM ADDANTEC))
COMPILE((ANDARGS ON2AVER MATCH1))
DEFINE(((MASK(LAMBDA(Z)(PROG(U U1 U2)
  (SETQ U W)
A  (COND((NULL U)(RETURN W))
    ((EQUAL Z (CADDDAAR U))(GO B)))
  (SETQ U1 (APPEND U1 (CONS(CAR U) NIL)))
  (SETQ U (CDR U))
  (GO A)
B  (COND((NULL U)(GO D))
    ((MEMBER(CAAAR U)(QUOTE(REL REL1)))(SETQ U2(APPEND U2 (CONS(CAR U)
       NIL))))
  (T (SETQ U1 (APPEND U1 (CONS(CAR U) NIL)))))
  (SETQ U (CDR U))
  (GO B)
D  (PRINT U2)
  (RETURN U1) )))))
COMPILE((MASK))
STOP)
```

```
                          GMATCH        LISP
DEFINE((IMPMATCH(LAMBDA(U M)(PROG(U1 U2 U3 U4)                                  00010
  (COND((EQUAL(CAR U)(QUOTE EQUAL))(GO A1))                                     00020
    ((EQUAL(CAR U)(QUOTE EQUAL2))(GO A2))                                       00030
    ((EQUAL(CAR U)(QUOTE EXISTS))(GO A8))                                       00040
    ((EQUAL(CAR U)(QUOTE IMPLIES))(GO A6))                                      00050
    ((EQUAL(CAR U)(QUOTE AND))(GO A3)))                                         00060
  (RETURN NIL)                                                                  00070
A1  (SETQ U2 (CONS(CADR U) NIL))                                                00080
  (GO A4)                                                                       00090
A2  (SETQ U2 (CDR U))                                                           00100
A4  (COND((NULL U2)(RETURN U1)))                                                00110
  (SETQ U3 (CAR U2))  (SETQ U2 (CDR U2))                                        00120
  (SETQ U4 (GMATCH U3 M))                                                       00130
  (COND((NOT(NULL U4))(SETQ U1 (APPEND U1 (CONS U4 NIL)))))                     00140
  (COND((EQUAL(CAR M)(QUOTE EXISTS))(GO A4))                                    00150
    ((MEMBER(CAR M)(QUOTE(EQUAL EQUAL2)))(SETQ U4 (MATCH1 U3 (CONS              00160
    (CADR M) NIL))))(T (SETQ U4 (MATCH1 U3 (CDR M)))))                          00170
  (COND((NOT(NULL U4))(SETQ U1 (APPEND U1 U4))))                                00180
  (GO A4)                                                                       00190
A3  (SETQ U2 (ANDARGS U))                                                       00200
  (GO A7)                                                                       00210
A6  (SETQ U2 (CDDR U))                                                          00220
  (SETQ U1 (ALLMATCH1(CAR U2) M))                                               00230
A7  (COND((NULL U2)(RETURN U1)))                                                00240
  (SETQ U3 (CAR U2))  (SETQ U2 (CDR U2))                                        00250
  (SETQ U1 (APPEND U1 (IMPMATCH U3 M)))                                         00260
  (GO A7)                                                                       00270
A8  (SETQ Z3 (SUBST(CONS(QUOTE RES)(CADR U))                                    00280
    (CONS N (CADR U)) Z3))                                                      00290
  (SETQ U1 (IMPMATCH(CADDR U) M))                                               00300
  (SETQ Z3 (SUBST(CONS N (CADR U))                                              00310
    (CONS(QUOTE RES)(CADR U)) Z3))                                              00320
  (RETURN U1) )))))                                                             00330
DEFINE((GMATCH1(LAMBDA(U U1 U2 V2)(PROG(V3 V4)                                  00340
  (COND((NULL U)(GO A1))                                                        00350
    ((NULL U1)(RETURN NIL))                                                     00360
    ((ATOM U)(GO A51))                                                          00370
    ((ATOM(CAR U))(GO A2))                                                      00380
    ((ATOM(CAR U1))(RETURN NIL)))                                               00390
  (SETQ V3 (GMATCH1(CAR U)(CAR U1) U2 V2))                                      00400
  (COND((NULL V3)(RETURN NIL)))                                                 00410
  (SETQ V4 (GMATCH1(CDR U)(CDR U1) U2 V2))                                      00420
  (COND((NULL V4)(RETURN NIL)))                                                 00430
  (RETURN(CONS(APPEND(CAR V3)(CAR V4))(APPEND(CDR V3)(CDR V4))))                00440
A1  (COND((NULL U1)(RETURN(CONS U2 V2))))                                       00450
  (RETURN NIL)                                                                  00460
A2  (COND((FREEVBLE(CAR U))(GO A3))                                             00470
    ((EQUAL(CAR U)(CAR U1))(GO A4))                                             00480
    ((AND(MEMBER(CAR U)(QUOTE(EQUAL2 EQUAL)))                                   00490
      (MEMBER(CAR U1)(QUOTE(EQUAL2 EQUAL))))(GO A4)))                           00500
  (RETURN NIL)                                                                  00510
A3  (SETQ V3 (GMATCH1 (CDR U)(CDR U1) U2 V2))                                   00520
  (COND((NULL V3)(RETURN NIL)))                                                 00530
  (RETURN(CONS(APPEND(CONS(CAR U)NIL)(CAR V3))(APPEND(CONS(CAR U1)              00540
    NIL)(CDR V3))))                                                             00550
A5  (COND((ON(CAR U1) Z3)(GO A5B)))                                             00560
  (RETURN NIL)                                                                  00570
A5B  (RETURN(CONS(CONS U U2)(CONS(CAR U1) V2)))                                 00580
A51  (COND((FREEVBLE U)(GO A5))                                                 00590
    ((EQUAL U (CAR U1))(RETURN(CONS U2 V2))))                                   00600
  (RETURN NIL)                                                                  00610
A4  (RETURN(GMATCH1 (CDR U)(CDR U1) U2 V2)) )))))                               00620
DEFINE((GMATCH(LAMBDA(U U1)(PROG(V7 V2 V3 V4 V5 V6)                             00630
  (SETQ V7 (GMATCH1 U U1 (CONS N NIL)(CONS N NIL)))                             00640
  (COND((NULL V7)(RETURN NIL)))                                                 00650
  (SETQ V2 (CDR V7))  (SETQ V7 (CAR V7))                                        00660
A1  (SETQ V3 (APPEND V3 (CONS(CAR V7) NIL)))                                    00670
  (SETQ V4 (APPEND V4 (CONS(CAR V2) NIL)))                                      00680
A1A  (SETQ V2 (CDR V2))                                                         00690
  (SETQ V7 (CDR V7))                                                            00700
  (COND((NULL V7)(RETURN (CONS V3 V4)))                                         00710
    ((MEMBER(CAR V7) V3)(GO A2)))                                               00720
  (GO A1)                                                                       00730
A2  (SETQ V5 V3)                                                                00740
  (SETQ V6 V4)                                                                  00750
A3  (COND((EQUAL(CAR V7)(CAR V5))(GO A4)))                                      00760
  (SETQ V5 (CDR V5))  (SETQ V6 (CDR V6))                                        00770
  (GO A3)                                                                       00780
A4  (COND((EQUAL(CAR V2)(CAR V6))(GO A1A)))                                     00790
  (RETURN NIL) )))))                                                            00800
COMPILE((IMPMATCH GMATCH1 GMATCH))  STOP)))                                     00810
```

```
                          VBLES        LISP
DEFINE(((VBLES(LAMBDA(U)(PROG()                                          00010
  (COND((NULL U)(RETURN NIL))                                            00020
    ((ATOM U)(RETURN(VBLES(CONS U NIL))))                                00030
    ((ATOM (CAR U))(GO A1)))                                             00040
  (RETURN(APPEND(VBLES(CAR U))(VBLES(CDR U))))                           00050
A1  (COND((ON(CAR U) Z1)(GO A2))                                         00060
    ((ON(CAR U) ZN)(GO A2))                                              00070
    ((ON(CAR U) Z2)(GO A2))                                              00080
    ((ON(CAR U) Z3)(GO A2)))                                             00090
  (RETURN (VBLES(CDR U)))                                                00100
A2  (RETURN(CONS(CAR U)(VBLES(CDR U))))  )))))                           00110
DEFINE(((ALLVBLES(LAMBDA(U)                                              00120
  (COND((NULL U) U)((MEMBER(CAR U)(CDR U))(ALLVBLES(CDR U)))             00130
    (T (CONS(CAR U)(ALLVBLES(CDR U))))) ))))                             00140
DEFINE(((RESRV(LAMBDA(U Z)                                               00150
  (COND((NULL Z) Z)((MEMBER(CDAR Z) U)                                   00160
    (CONS(CONS(QUOTE RES)(CDAR Z))                                       00170
    (RESRV U (CDR Z))))                                                  00180
    (T (CONS(CAR Z)(RESRV U (CDR Z))))))  ))))                           00190
DEFINE(((ON(LAMBDA(U Z)                                                  00200
  (COND((NULL Z) NIL)((EQUAL U (CDAR Z)) T)                              00210
    (T (ON U (CDR Z))))) ))))                                            00220
DEFINE(((CONNECTIVE(LAMBDA(U)                                            00230
  (COND((MEMBER U (QUOTE(IMPLIES IMPLIES2 OR DEFER                       00240
    EQUAL FEQUAL EQUAL2 EXISTS AND NOT))) T)                             00250
  (T F)) ))))                                                            00260
DEFINE(((TERMS(LAMBDA(U L)(PROG()                                        00270
  (COND((NULL U)(RETURN NIL))                                            00280
    ((ATOM(CAR U))(GO A1)))                                              00290
  (RETURN (APPEND(TERMS(CAR U) L)(TERMS(CDR U) L)))                      00300
A1  (COND((CONNECTIVE(CAR U))(GO A2))                                    00310
    ((EQUAL(CAR U)(QUOTE ASSOC))(GO A2))                                 00320
    ((NUMBERP(CAR U))(GO A2))                                            00330
    ((MEMBER(CAR U) L)(GO A2)))                                          00340
  (RETURN(CONS U (TERMS(CDR U) L)))                                      00350
A2  (RETURN (TERMS (CDR U) L)) )))))                                     00360
DEFINE(((GENSUBST(LAMBDA(L M U)(PROG()                                   00370
  (COND((NULL L)(RETURN U))                                              00380
    ((ATOM(CAR L))(GO A))                                                00390
    ((ON(CAAR L) Z2)(GO B)))                                             00400
C   (RETURN(SUBST(CAR L)(CAR M)(GENSUBST(CDR L)(CDR M) U)))              00410
A   (COND((EQUAL(CAR L)(CAR M))(RETURN(GENSUBST(CDR L)                   00420
    (CDR M) U))))                                                        00430
  (GO C)                                                                 00440
B   (COND((ATOM(CAR M))(GO C))                                           00450
    ((ON(CAAR M) Z2)(RETURN(GENSUBST(CAR L)                              00460
    (CAR M)(GENSUBST(CDR L)(CDR M) U))))                                 00470
  (GO C) )))))                                                           00480
COMPILE((VBLES ALLVBLES RESRV ON))                                       00490
COMPILE((CONNECTIVE TERMS GENSUBST))                                     00500
DEFINE(((SUPXEC(LAMBDA(U6 U1)(PROG(SUP RET U1)                           00510
A   (COND((NULL U6)(RETURN RET)))                                        00520
  (SETQ SUP NIL)                                                         00530
  (SETQ RET (CONS(SUPSUB U6 U) RET))                                     00540
B   (COND((NULL U6)(GO B1))                                             00550
    ((NOT(MEMBER(CDAR U6) SUP))(SETQ U1 (CONS(CAR U6) U1))))             00560
  (SETQ U6 (CDR U6))                                                     00570
  (GO B)                                                                 00580
B1  (SETQ U6 U1)                                                         00590
  (SETQ U1 NIL)                                                          00600
  (GO A) )))))                                                           00610
DEFINE(((SUPSUB(LAMBDA(U6 U)(PROG(U3)                                    00620
  (SETQ U3 U6)                                                           00630
A1  (COND((NULL U3)(GO B1))                                              00640
    ((EQUAL(CDAR U3) U)(GO B2))                                          00650
    (T (SETQ U3 (CDR U3))))                                              00660
  (GO A1)                                                                00670
B1  (COND((ATOM U)(RETURN U)))                                           00680
  (RETURN(CONS(SUPSUB U6 (CAR U))(SUPSUB U6 (CDR U))))                   00690
B2  (SETQ SUP (CONS(CDAR U3) SUP))                                       00700
  (RETURN (CAAR U3)) )))))                                               00710
STOP))))                                                                 00720
```

```
DEFINE(((NOLOOK(LAMBDA(M)                                                    00010
(COND((MEMBER(CAAR(GFIND M))(QUOTE(VER IRR VERA2 NILL)))                     00020
T)(T F)) ))))                                                                00030
DEFINE(((SPREAD(LAMBDA(A)(UNPACK(CAR(GET A (QUOTE PNAME)))))))               00040
DEFINE(((ASTERISKED(LAMBDA(A)(COND((EQUAL STAR                               00050
(CAR(SPREAD A))) T)(T F)) ))))                                               00060
DEFINE(((FIFTH(LAMBDA(U)(CAR(CDDDDR U))))))                                 00070
DEFINE(((SIXTH(LAMBDA(U)(COND((NULL(CDR(CDDDDR U))) NIL)                    00080
  (T (CADR(CDDDDR U)))) ))))                                                 00090
DEFINE(((ALLSUBSTS(LAMBDA(L M U LN)(PROG(U1 U2 S)                            00100
  (SETQ U1 (GMATCH M U))                                                     00110
  (COND((NOT(NULL U1))(SETQ U1 (CONS U1 NIL))))                              00120
A1  (COND((NULL U1)(GO A2)))                                                 00130
  (SETQ U2 (APPEND U2 (LIST(CONS(GENSUBST(CDAR U1)                           00140
    (CAAR U1) L)(GENSUBST(CDAR U1)(CAAR U1) M)))))                           00150
  (SETQ U1 (CDR U1))                                                         00160
  (GO A1)                                                                    00170
A2  (COND((NOT(NULL S))(GO A3)))                                             00180
  (SETQ S N)                                                                 00190
  (COND((MEMBER(CAR U)(QUOTE(EQUAL EQUAL2)))(SETQ U (CONS(CADR U) NIL)))     00200
    (T (SETQ U (CDR U))))                                                    00210
  (SETQ U1 (MATCH1 M U))                                                     00220
  (GO A1)                                                                    00230
A3  (COND((NULL U2)(RETURN NIL)))                                            00240
  (RETURN(CONS LN U2)) )))))                                                 00250
DEFINE(((ASSOCM(LAMBDA(M U LN)(PROG(U1)                                      00260
  (COND((MEMBER(CAR U)(QUOTE(EQUAL EQUAL2)))(SETQ U (CADR U))))              00270
  (SETQ U1 (PMATCH M U))                                                     00280
  (COND((NULL U1)(RETURN NIL)))                                              00290
  (RETURN(CONS LN U1)) )))))                                                 00300
DEFINE(((PMATCH(LAMBDA(M U)(PROG(U1 U2 U3 S)                                 00310
  (SETQ U1 (CONS(QUOTE(*PROD A1 (*PROD A2 A3 A) A))                          00320
    (QUOTE(*PROD(*PROD A1 A2 A) A3 A))))                                     00330
  (SETQ U1 (SUBST M (QUOTE A) U1))                                           00340
  (SETQ U2 (RPLCE(QUOTE(A1 A2 A3)) NIL))                                     00350
  (COND((NOT(NULL U2)))(SETQ U1 (GENSUBST(CAR U2)(CDR U2) U1))))             00360
C  (SETQ U2 (MATCH1(CDR U1) U))                                              00370
A  (COND((NULL U2)(GO B)))                                                   00380
  (SETQ U3 (APPEND U3 (LIST(GENSUBST(CDAR U2)(CAAR U2) U1))))                00390
  (SETQ U2 (CDR U2))                                                         00400
  (GO A)                                                                     00410
B  (COND((NOT(NULL S))(RETURN U3)))                                          00420
  (SETQ S N)                                                                 00430
  (SETQ U1 (CONS(CDR U1)(CAR U1)))                                           00440
  (GO C) )))))                                                               00450
DEFINE(((ROFV(LAMBDA(Z FLG)(PROG(U U1)                                       00460
  (COND((EQUAL Z N)(RETURN VV2)))                                            00470
  (SETQ U VV2)                                                               00480
A  (COND((NULL U)(RETURN U1))                                                00490
    ((MEMBER(CAAAR U) FLG)(SETQ U1(APPEND U1 (CONS(CAR U) NIL))))            00500
    ((LESSP(CAAAR U) Z)(GO B))                                               00510
    (T (RETURN(APPEND U1 U))))                                               00520
B  (SETQ U (CDR U))                                                          00530
  (GO A) )))))                                                               00540
DEFINE(((CHNGW(LAMBDA(U)(PROG(U1 U2 U3 U4)                                   00550
  (COND((EQUAL(FIFTH(CAR U)) Z4A)(RETURN NIL)))                              00560
  (SETQ U1 W)                                                                00570
A1  (COND((NULL U1)(RETURN NIL))                                             00580
    ((EQUAL(CDAR U1)(CDR U))(GO A2)))                                        00590
  (SETQ U2 (APPEND U2 (CONS(CAR U1) NIL)))                                   00600
  (SETQ U1 (CDR U1))                                                         00610
  (GO A1)                                                                    00620
A2  (SETQ U3 (CAAR U1))                                                      00630
  (SETQ U4 (SIXTH U3))                                                       00640
  (SETQ U3 (LIST(CAR U3)(CADR U3)(CADDR U3)(CADDDR U3) Z4A))                 00650
  (COND((NOT(NULL U4)))(SETQ U3 (APPEND U3 (CONS U4 NIL)))))                 00660
  (SETQ W (APPEND U2 (CONS(CONS U3 (CDR U))(CDR U1))))                       00670
  (RETURN NIL) )))))                                                         00680
STOP)))))                                                                    00690
```

```
                              NEXT        LISP
DEFINE((((NEXT(LAMBDA(L)(PROG()                                                  00010
  (COND((NOT(EQUAL(CAAAR W)(QUOTE PNT)))(GO B1)))                                00020
  (SETQ W1 (APPEND W1 (CONS(SUBST(QUOTE REL)                                     00030
    (QUOTE PNT)(CAR W)) NIL)))                                                   00040
B  (SETQ W (CDR W))                                                              00050
  (COND((NULL W)(RETURN NIL))                                                    00060
    ((EQUAL(CAAAR W)(QUOTE REL))(GO C)))                                         00070
B1  (SETQ W1 (APPEND W1 (CONS(CAR W) NIL)))                                      00080
  (GO B)                                                                         00090
C  (COND((EQUAL(CADAR W)(QUOTE DEFER))(GO D)))                                  00100
C1  (SETQ W (CONS(SUBST(QUOTE PNT)(QUOTE REL)(CAR W))(CDR W)))                   00110
  (RETURN N)                                                                     00120
D  (SETQ L (CDAAR W))                                                            00130
  (COND((MEMBER(CADDR L)(CADDAR(GFIND(CAAR L))))(GO C1))                         00140
    ((NULL(CDR W))(RETURN NIL)))                                                 00150
  (SETQ W1 (SUBST Z4 (CADDR L) W1))                                              00160
  (SETQ W (APPEND W (CONS(SUBST Z4 (CADDR L)(CAR W)) NIL)))                      00170
  (SETQ Z4 (ADD1 Z4))                                                            00180
  (GO B) )))))                                                                   00190
DEFINE((((CADDDAR(LAMBDA(U)(CAR(CDDDAR U))))))                                  00200
DEFINE((((CADDDAAR(LAMBDA(U)(CADR(CDDAAR U))))))                                00210
DEFINE((((CADDAAR(LAMBDA(U)(CAR(CDDAAR U))))))                                  00220
DEFINE((((RPLCE(LAMBDA(U U1)(PROG(U2 U3 U4 U5 U6)                               00230
  (SETQ U2 U)                                                                    00240
A1  (COND((NULL U2)(GO A6))                                                      00250
    ((FREEVBLE(CAR U2))(GO A3)))                                                 00260
A2  (COND((ON(CAR U2) Z1)(SETQ U5 Z1))                                           00270
    ((ON(CAR U2) ZN)(SETQ U5 ZN))                                                00280
    ((ON(CAR U2) Z2)(SETQ U5 Z2))                                                00290
    (T (SETQ U5 Z3)))                                                            00300
  (SETQ U6 (CAR U5))                                                             00310
A2C  (COND((NULL U5)(GO A5))                                                     00320
    ((EQUAL N (CAAR U5))(GO A4)))                                                00330
A2D  (SETQ U5 (CDR U5))                                                          00340
  (GO A2C)                                                                       00350
A3  (COND((MEMBER(CAR U2) U1)(GO A2)))                                           00360
A3A  (SETQ U2 (CDR U2))                                                          00370
  (GO A1)                                                                        00380
A4  (COND((MEMBER(CDAR U5) U1)(GO A2D))                                          00390
    ((MEMBER(CDAR U5) U)(GO A2D))                                                00400
    ((MEMBER(CDAR U5) U3)(GO A2D)))                                              00410
  (SETQ U3 (CONS(CDAR U5) U3))                                                   00420
  (SETQ U4 (CONS(CAR U2) U4))                                                    00430
  (GO A3A)                                                                       00440
A5  (PRINT(QUOTE(NOT ENOUGH VARIABLES)))                                         00450
  (PRINT U6)                                                                     00460
  (SETQ U (ROFLX()))                                                             00470
  (SETQ U5 (CONS(CONS N U) NIL))                                                 00480
  (COND((EQUAL U6 (CAR Z1))(SETQ Z1 (APPEND Z1 U5)))                             00490
    ((EQUAL U6 (CAR ZN))(SETQ ZN (APPEND ZN U5)))                                00500
    ((EQUAL U6 (CAR Z2))(SETQ Z2 (APPEND Z2 U5)))                                00510
    (T (SETQ Z3 (APPEND Z3 U5))))                                                00520
  (GO A4)                                                                        00530
A6  (COND((NULL U3)(RETURN NIL)))                                                00540
  (RETURN(CONS U3 U4)) )'))                                                      00550
DEFINE((((ARGOF1(LAMBDA(U L)(PROG(U1)                                            00560
  (COND((NULL L)(RETURN NIL))                                                    00570
    ((CONNECTIVE(CADAR L))(RETURN(ARGOF1 U (CDR L))))                            00580
    ((ASTERISKED(CADAR L))(RETURN(ARGOF1 U (CDR L))))                            00590
    ((ON U Z2)(GO A))                                                            00600
    ((EQUAL U (CADDAR L))(RETURN(CONS(CDAR L)                                    00610
      (ARGOF1 U (CDR L))))))                                                     00620
  (RETURN(ARGOF1 U (CDR L)))                                                     00630
A  (COND((ATOM(CADDAR L))(RETURN(ARGOF1 U (CDR L))))                             00640
    ((EQUAL U (CAR(CADDAR L)))(RETURN(CONS(CDAR L)                               00650
    (ARGOF1 U (CDR L))))))                                                       00660
  (RETURN(ARGOF1 U (CDR L))) )))))                                               00670
DEFINE((((ROFW(LAMBDA(M)(PROG(U)                                                 00680
  (SETQ U W)                                                                     00690
A  (COND((NULL U)(RETURN NIL))                                                   00700
    ((GREATERP M (CADDDAAR U))(SETQ U (CDR U)))                                  00710
    (T (RETURN U)))                                                              00720
  (GO A) )))))                                                                   00730
COMPILE((NEXT CADDDAR CADDDAAR CADDAAR RPLCE ARGOF1 ROFW))                       00740
STOP)                                                                            00750
```

```
                                ATMS        LISP
DEFINE(((GENVBLE(LAMBDA(U)(PROG(U2 U3)                                          00010
  (SETQ U2 Z3)                                                                  00020
A  (COND((NULL U2)(GO B))                                                       00030
    ((EQUAL(CAAR U2)(QUOTE GEN))(SETQ U3 (CONS(CDAR U2) U3))))                  00040
  (SETQ U2 (CDR U2))                                                            00050
  (GO A)                                                                        00060
B  (SETQ U2 (ATMS U))                                                           00070
C  (COND((NULL U2)(RETURN NIL))                                                00080
    ((MEMBER(CAR U2) U3)(RETURN(CAR U2))))                                      00090
  (SETQ U2 (CDR U2))                                                            00100
  (GO C) )))))                                                                  00110
DEFINE(((ATMS(LAMBDA(U)(COND((NULL U) NIL)((ATOM U)                             00120
(CONS U NIL))(T (APPEND(ATMS(CAR U))(ATMS(CDR U)))))))))                        00130
DEFINE(((RPLCEF(LAMBDA(U U1)(PROG(U2 U3)                                        00140
  (SETQ U2 Z3)                                                                  00150
A  (COND((EQUAL(CDAR Z3)(QUOTE A1))(GO B)))                                     00160
  (SETQ Z3 (CDR Z3))                                                            00170
  (GO A)                                                                        00180
B  (SETQ U3 (RPLCE U U1))                                                       00190
  (SETQ Z3 U2)                                                                  00200
  (RETURN U3) )))))                                                             00210
DEFINE(((FREEVBLE(LAMBDA(U)(PROG(U1)                                            00220
  (SETQ U1 (APPEND Z1 (APPEND Z2 (APPEND Z3 ZN))))                              00230
A1  (COND((NULL U1)(RETURN NIL))                                                00240
    ((EQUAL U (CDAR U1))(GO A2)))                                               00250
  (SETQ U1 (CDR U1))                                                            00260
  (GO A1)                                                                       00270
A2  (COND((EQUAL(CAAR U1) N)(RETURN T)))                                        00280
  (RETURN NIL) )))))                                                            00290
DEFINE(((TACK(LAMBDA(L M)(PROG(U1)                                              00300
  (SETQ U1 (CONS(CONS(CAR L) M)NIL))                                            00310
A1  (SETQ L (CDR L))                                                            00320
  (COND((NULL L)(RETURN U1)))                                                   00330
  (SETQ U1 (APPEND U1 (CONS(CONS(CAR L) M) NIL)))                               00340
  (GO A1) )))))                                                                 00350
DEFINE(((MIRGEA(LAMBDA(L M)(PROG()                                              00360
A1  (COND((NULL L)(RETURN M))                                                   00370
    ((EQUAL(CAAR L)(QUOTE HEAD))(GO A2))                                        00380
    ((EQUAL(CAAR L)(QUOTE NONE))(GO A2))                                        00390
    ((ATOM(CAAR L))(GO A3)))                                                    00400
  (SETQ L (CONS(CONS(CAAAR L)(CDAR L))(CONS(CONS(CDAAR L)                       00410
    (CDAR L))(CDR L))))                                                         00420
  (GO A1)                                                                       00430
A3  (SETQ M (APPEND M (CONS (CAR L) NIL)))                                      00440
A2  (SETQ L (CDR L))                                                            00450
  (GO A1) )))))                                                                 00460
DEFINE(((DELPR(LAMBDA(M L)(PROG(U1)                                             00470
A1  (COND((NULL L)(RETURN U1))                                                  00480
    ((ATOM(CAR L))(GO A2))                                                      00490
    ((EQUAL M (CAAR L))(GO A4))                                                 00500
    ((EQUAL M (CDAR L))(GO A3)))                                                00510
A2  (SETQ U1 (APPEND U1 (CONS(CAR L) NIL)))                                     00520
  (SETQ L (CDR L))                                                              00530
  (GO A1)                                                                       00540
A3  (SETQ L (CONS(CAAR L)(CDR L)))                                              00550
  (RETURN (APPEND U1 L))                                                        00560
A4  (SETQ L (CONS(CDAR L)(CDR L)))                                              00570
  (RETURN (APPEND U1 L)) )))))                                                  00580
DEFINE(((ONLYMEMB(LAMBDA(M L)                                                   00590
  (COND((NULL(CDR L))(COND((EQUAL M (CAR L)) T)(T F)))                          00600
    ((EQUAL(CADR L)(QUOTE HEAD))(COND((EQUAL M (CAR L)) T)(T F)))               00610
    ((EQUAL(CAR L)(QUOTE HEAD))(COND((EQUAL M (CADR L)) T)(T F)))               00620
    (T F)) ))))                                                                 00630
DEFINE(((DELTF(LAMBDA(M L)(PROG(U1)                                             00640
A1  (COND((EQUAL(CAR L) M)(RETURN (APPEND U1 (CDR L)))))                        00650
  (SETQ U1 (APPEND U1 (CONS (CAR L) NIL)))                                      00660
  (SETQ L (CDR L))                                                              00670
  (GO A1) )))))                                                                 00680
COMPILE((FREEVBLE TACK MIRGEA DELPR ONLYMEMB DELTF))                            00690
STOP)                                                                           00700
```

```
                          GFIND        LISP
DEFINE(((ANDEND(LAMBDA(M U)(PROG(U1)                                          00010
A1  (COND((NULL U)(RETURN N))                                                 00020
    ((ATOM(CAR U))(GO A2)))                                                   00030
A1A  (SETQ U (CDR U))                                                         00040
  (GO A1)                                                                     00050
A2  (COND((EQUAL(CAR U)(QUOTE NONE))(GO A1A)))                                00060
  (SETQ U1 (GFIND(CAR U)))                                                    00070
  (COND((EQUAL(CADR U1)(QUOTE AND))(GO A3)))                                  00080
  (GO A1A)                                                                    00090
A3  (COND((MEMBER M (ANDTERMS(CADDAR U1)))(RETURN NIL)))                      00100
  (GO A1A) )))))                                                              00110
DEFINE(((ANDTERMS(LAMBDA(U)(PROG(U1 U2)                                       00120
  (COND((ATOM(CAR U))(GO B)))                                                 00130
  (SETQ U1 (GFIND(CAAR U)))                                                   00140
  (COND((EQUAL(CADR U1)(QUOTE AND))(SETQ U2 (ANDTERMS(CADDAR U1))))          00150
    (T (SETQ U2 (CONS(CAAR U) NIL))))                                         00160
A2  (SETQ U1 (GFIND(CDAR U)))                                                 00170
  (COND((EQUAL(CADR U1)(QUOTE AND))(GO A3)))                                  00180
  (RETURN(CONS(CDAR U) U2))                                                   00190
A3  (RETURN(APPEND(ANDTERMS(CADDAR U1)) U2))                                  00200
B  (SETQ U1 (GFIND(CAR U)))                                                   00210
  (COND((EQUAL(CADR U1)(QUOTE AND))(RETURN(ANDTERMS(CADDAR U1))))))          00220
  (RETURN(CONS(CAR U) NIL)) )))))                                             00230
DEFINE(((GFIND(LAMBDA(M)(PROG(U)                                              00240
  (SETQ U (APPEND W1 W))                                                      00250
A1  (COND((NULL U)(RETURN NIL))                                               00260
    ((ATOM M)(GO B))                                                          00270
    ((EQUAL M (CDAR U))(RETURN(CAR U))))                                      00280
A2  (SETQ U (CDR U))                                                          00290
  (GO A1)                                                                     00300
B  (COND((EQUAL M (CADDDAAR U))(RETURN(CAR U))))                              00310
  (GO A2) )))))                                                               00320
COMPILE((ANDEND ANDTERMS GFIND))                                              00330
DEFINE(((ALLMATCH1(LAMBDA(U M)(PROG(U1 U3)                                    00340
  (COND((EQUAL(CAR U)(QUOTE AND))(SETQ U (CONS U (ANDARGS U))))               00350
    (T (SETQ U (CONS U NIL))))                                                00360
B  (COND((NULL U)(RETURN U1)))                                                00370
  (SETQ U3 (GMATCH(CAR U) M))                                                 00380
  (COND((NOT(NULL U3))(SETQ U1 (APPEND U1 (CONS U3 NIL)))))                   00390
  (SETQ U (CDR U))                                                            00400
  (GO B) )))))                                                                00410
DEFINE(((GMEMB(LAMBDA(M U)                                                    00420
  (COND((NULL U) F)                                                           00430
    ((ATOM(CAR U))(COND((EQUAL M (CAR U)) T)                                  00440
      (T (GMEMB M (CDR U)))))                                                 00450
    (T (COND((EQUAL M (CAAR U)) T)                                            00460
      ((EQUAL M (CDAR U)) T)                                                  00470
      (T (GMEMB M (CDR U))))))) ))))                                          00480
DEFINE(((RPLC1(LAMBDA(U)(PROG(U1 U2 U3)                                       00490
A0  (COND((NULL U)(GO B)))                                                    00500
  (SETQ U3 (APPEND Z1 (APPEND Z2 (APPEND Z3 ZN))))                            00510
A1  (COND((EQUAL(CAR U)(CDAR U3))(GO A3)))                                    00520
  (SETQ U3 (CDR U3))                                                          00530
  (GO A1)                                                                     00540
A3  (COND((EQUAL(CAAR U3)(QUOTE RES))(GO A4)))                                00550
  (SETQ U3 (GENSYM))                                                          00560
  (SETQ U1 (CONS U3 U1))                                                      00570
  (SETQ U2 (CONS(CAR U) U2))                                                  00580
  (COND((ON(CAR U) Z1)(SETQ Z1 (CONS(CONS(QUOTE RES) U3) Z1)))                00590
    ((ON(CAR U)ZN)(SETQ ZN (CONS(CONS(QUOTE RES) U3) ZN)))                    00600
    ((ON(CAR U) Z2)(SETQ Z2 (CONS(CONS(QUOTE RES) U3) Z2)))                   00610
    (T (SETQ Z3 (CONS(CONS(QUOTE RES) U3) Z3))))                              00620
A4  (SETQ U (CDR U))                                                          00630
  (GO A0)                                                                     00640
B  (COND((NULL U1)(RETURN NIL)))                                              00650
  (RETURN(CONS U1 U2)) )))))                                                  00660
COMPILE((ALLMATCH1 GMEMB RPLC1))                                              00670
STOP)                                                                         00680
```

```
                    DEFNS        DATA
{                                                                                                                    00010
((SUBGROUP A B))((SNORMAL A B C))                                                                                    00020
((MEMBER A1 A))((NORMAL A B C))((SEMIGROUP A))((ASSOC A))((ABELIAN A B))                                             00030
((SUBSET A B))                                                                                                       00040
((SUBGROUP A B) IMPLIES(AND(MEMBER A1 A)(MEMBER A2 A))                                                               00050
(AND(MEMBER(*PROD A1 A2 B) A)(MEMBER(*INVERSE A1 B) A)))                                                             00060
((SUBMONOID A B) IMPLIES(AND(MEMBER A1 A)(MEMBER A2 A))                                                              00070
(MEMBER(*PROD A1 A2 B) A))                                                                                           00080
((HOMOMORPHISM(F1 A B)) IMPLIES(AND(MEMBER A1 A)                                                                     00090
(MEMBER A2 A))(AND(MEMBER(F1 A B A1) B)(EQUAL2                                                                       00100
(F1 A B (*PROD A1 A2 A))(*PROD(F1 A B A1)(F1 A B A2) B))))                                                           00110
((HOMOMF(F1 A B)) IMPLIES(AND(MEMBER A1 A)(MEMBER A2 A))                                                             00120
(FEQUAL(F1 A B (*PROD A1 A2 A))(*PROD(F1 A B A1)                                                                     00130
(F1 A B A2) B)))                                                                                                     00140
((ONTO(F1 A B)) IMPLIES(MEMBER A1 B)(EXISTS A2                                                                       00150
(AND(MEMBER A2 A)(EQUAL(F1 A B A2) A1))))                                                                            00160
((INVERSE A1 A2 A) EQUAL(*PROD A1 A2 A)(*IDENTITY A))                                                                00170
((SHAS*COMP*SERIES A) EXISTS N1 (EXISTS(B N2)                                                                        00180
(COMP*SERIES((B N2) N1) A)))                                                                                         00190
((ONETONE(F1 A B)) IMPLIES(AND(AND(MEMBER A1 A)                                                                      00200
(MEMBER A2 A))(EQUAL(*PROD(*INVERSE(F1 A B A1) B)                                                                    00210
(F1 A B A2) B)(*IDENTITY B)))(EQUAL(*PROD                                                                            00220
(*INVERSE A1 A) A2 A)(*IDENTITY A)))                                                                                 00230
((RCOSET A A1 B C))((LCOSET A A1 B C))                                                                               00240
                                                                                                                     00250
){                                                                                                                   00260
((SUBGROUP A B))((SNORMAL A B C))((NULL))                                                                            00270
((*FININT A N9) AND(EQUAL(*FININT A 1)(A 1))                                                                         00280
(EQUAL(*FININT A (*SUCCESSOR N3))(*INT(*FININT A N3)                                                                 00290
(A (*SUCCESSOR N3)))))                                                                                               00300
((*SEQ(A1 N) A))((*EXP A1 N A))                                                                                      00310
((*SUCCESSOR N))                                                                                                     00320
((MEMBER A1 A))((*PROD A1 A2 A))((ASSOC A))                                                                          00330
((IMAGE A (F1 B C)) IMPLIES2(MEMBER A1 A)(EXISTS A2                                                                  00340
(AND(MEMBER A2 B)(SEQUAL(F1 B C A2) A1))))                                                                           00350
((HOMOMORPHISM(F1 A B)) IMPLIES(MEMBER A1 A)                                                                         00360
(MEMBER(F1 A B A1) B))                                                                                               00370
((*INVERSE A1 A)) ((*IDENTITY A))                                                                                    00380
((GROUP A) AND(IMPLIES(AND(MEMBER A1 A)                                                                              00390
(MEMBER A2 A))(MEMBER(*PROD A1 A2 A) A))(AND                                                                         00400
(ASSOC A)(AND(MEMBER(*IDENTITY A) A)                                                                                 00410
(IMPLIES(MEMBER A1 A)(MEMBER(*INVERSE A1 A) A)))))                                                                   00420
((CONJUGATE A A1 B C) AND(MEMBER A1 C)                                                                               00430
(IMPLIES2(MEMBER A2 A)(EXISTS A3 (AND(MEMBER A3 B)                                                                   00440
(EQUAL A2 (*PROD(*PROD A1 A3 C)(*INVERSE A1 C) C))))))                                                               00450
((SUBGROUP A B) AND(IMPLIES(MEMBER A1 A)                                                                             00460
(MEMBER A1 B))(AND(IMPLIES(AND(MEMBER A1 A)                                                                          00470
(MEMBER A2 A))(MEMBER (*PROD A1 A2 B) A))                                                                            00480
(AND(MEMBER(*IDENTITY B) A)(IMPLIES(MEMBER A1 A)                                                                     00490
(MEMBER(*INVERSE A1 B) A)))))                                                                                        00500
((KERNEL A (F1 B C))                                                                                                 00510
IMPLIES2(MEMBER A1 A)(EQUAL(F1 B C A1)(*IDENTITY C)))                                                                00520
((ABELIAN A B) IMPLIES(AND(MEMBER A1 A)(MEMBER A2 A))                                                                00530
(EQUAL(*PROD A1 A2 B)(*PROD A2 A1 B)))                                                                               00540
((SEMIGROUP A) AND(IMPLIES(AND(MEMBER A1 A)                                                                          00550
(MEMBER A2 A))(MEMBER(*PROD A1 A2 A) A))(ASSOC A))                                                                   00560
((CENTER A B) AND(IMPLIES(MEMBER A1 A)(MEMBER A1 B))                                                                 00570
(IMPLIES2(MEMBER A1 A)(IMPLIES(MEMBER A2 B)                                                                          00580
(EQUAL(*PROD A1 A2 B)(*PROD A2 A1 B)))))                                                                             00590
((INTERSECTION A B C) IMPLIES2(MEMBER A1 A)(AND                                                                      00600
(MEMBER A1 B)(MEMBER A1 C)))                                                                                         00610
((NORMAL A B C) IMPLIES(AND(MEMBER A1 B)(MEMBER A2 A))                                                               00620
(MEMBER(*PROD(*PROD A1 A2 C)(*INVERSE A1 C) C) A))                                                                   00630
((COMMUTATOR A1 A2 A3 A) EQUAL2 A1 (*PROD(*PROD(*PROD                                                                00640
(*INVERSE A2 A)(*INVERSE A3 A ) A) A2 A) A3 A))                                                                      00650
((NORMALIZER A B C) IMPLIES2(MEMBER A1 A)(IMPLIES                                                                    00660
(MEMBER A3 B)(MEMBER(*PROD(*PROD A1 A3 C)                                                                            00670
(*INVERSE A1 C) C) B)))                                                                                              00680
((RIMAGE A B (F1 C D)) IMPLIES2(MEMBER A1 A)(EXISTS A2                                                               00690
(AND(MEMBER A2 B)(SEQUAL(F1 C D A2) A1))))                                                                           00700
((EPIMORPHISM(F1 A B)) AND(IMPLIES(MEMBER A1 B)(EXISTS A2 (AND                                                       00710
(MEMBER A2 A)(EQUAL2(F1 A B A2) A1))))(IMPLIES(MEMBER A3 A)                                                          00720
(MEMBER(F1 A B A3) B)))                                                                                              00730
((MONOMORPHISM(F1 A B)) AND(ONEONE(F1 A B))                                                                          00740
(IMPLIES(MEMBER A1 A)(MEMBER(F1 A B A1) B)))                                                                         00750
((RCOSET A A1 B C) AND(MEMBER A1 C)(IMPLIES2(MEMBER A2 A)                                                            00760
(EXISTS A3 (AND(MEMBER A3 B)(SEQUAL(*PROD A3 A1 C) A2)))))                                                           00770
((LCOSET A A1 B C) AND(MEMBER A1 C)(IMPLIES2(MEMBER A2 A)                                                            00780
(EXISTS A3 (AND(MEMBER A3 B)(SEQUAL(*PROD A1 A3 C) A2)))))                                                           00790
((SETINV A B C) IMPLIES2(MEMBER A1 A)                                                                                00800
(EXISTS A2,(AND(MEMBER A2 B)(SEQUAL(*INVERSE A2 C) A1))))
```

```
((FACTORGROUP A B D) AND(AND                                                    00810
(IMPLIES2(MEMBER(*LCOSET A1 B D) A)(MEMBER A1 D))                               00820
(IMPLIES(MEMBER A1 B)(EQUAL(*LCOSET A1 B D)                                     00830
(*IDENTITY A))))(IMPLIES(MEMBER A1 A)                                           00840
(EXISTS A2 (AND(MEMBER A2 D)(EQUAL A1                                           00850
(*LCOSET A2 B D))))))                                                           00860
((UNITSET A A1) IMPLIES2(MEMBER A2 A)(SEQUAL A2 A1))                           00870
((INVIMAGE A B (F1 C D)) IMPLIES2(MEMBER A1 A)(MEMBER(F1 C D A1) B))            00880
((SUBSET A B) IMPLIES(MEMBER A1 A)(MEMBER A1 B))                                00890
((ISOMORPHISM(F1 A B)) AND(AND(ONEONE(F1 A B))                                  00900
(IMPLIES(MEMBER A1 B)(EXISTS A2 (AND                                            00910
(MEMBER A2 A)(EQUAL2(F1 A B A2) A1)))))(IMPLIES(MEMBER A3 A)                    00920
(MEMBER(F1 A B A3) B)))                                                         00930
((SUBFGRP A B C D) AND                                                          00940
(IMPLIES2(MEMBER(*LCOSET A1 B D) A)(MEMBER A1 C))                               00950
(IMPLIES(MEMBER A1 A)                                                           00960
(EXISTS A2 (AND(MEMBER A2 C)(EQUAL A1                                           00970
(*LCOSET A2 B D))))))                                                           00980
((COMP*SERIES((A N1) N2) B) AND(IMPLIES(EQUAL B (*UNITSET                       00990
(*IDENTITY B)))(COMP*SERIES((B N1) 1) B))(IMPLIES(AND(                          01000
$HAS*MAX*NORM*SUB B)(DEFER(EXISTS N4 (EXISTS(C N5) (COMP*SERIES                 01010
((C N5) N4)(*MAX*NORM*SUB B))))))                                               01020
(AND(COMP*SERIES((D N3) N6) B)(AND(AND(EACH(EQUAL(D N3)(C N5))                  01030
(*PREDECESSOR N6))(EQUAL(D N6) B))(EQUAL(*PREDECESSOR N6) N4)))))               01040
((ONETOONE(F1 A B)) IMPLIES(FEQUAL(F1 A B A1)                                   01050
(F1 A B A2))(EQUAL A1 A2))                                                      01060
((INVPNT A A1 (F1 B C)) AND(IMPLIES(MEMBER A2 A)                                01070
(EQUAL(F1 B C A2) A1))(IMPLIES(EQUAL(F1 B C A2)                                 01080
A1)(MEMBER A2 A)))                                                              01090
((CENTRAL A B C) AND(IMPLIES(MEMBER A1 A)(MEMBER A1 C)                          01100
)(IMPLIES2(MEMBER A1 A)(IMPLIES(MEMBER A2 B)                                    01110
(EQUAL(*PROD A1 A2 C)(*PROD A2 A1 C)))))                                        01120
)(                                                                              01130
((HMPRP(F1 A B)) (F1 A B (*IDENTITY A))(*IDENTITY B)                            01140
(F1 A B (*INVERSE A1 A))(*INVERSE(F1 A B A1) B)                                 01150
(F1 A B (*PROD A1 A2 A))(*PROD(F1 A B A1)                                       01160
(F1 A B A2) B))                                                                 01170
((FGPRP A B D)(*LCOSET(*INVERSE A1 D) B D)                                      01180
(*INVERSE(*LCOSET A1 B D) A)(*LCOSET(*PROD                                      01190
A1 A2 D) B D)(*PROD(*LCOSET A1 B D)(*LCOSET A2 B D) A))                          01200
)(                                                                              01210
((SUBSET A B)((A B))(A B))                                                      01220
((FACTORGROUP A B D)((A)(B D))(B D))                                            01230
((HOMOMORPHISM(F1 A B))((A)(B)))                                                01240
((EPIMORPHISM(F1 A B))((A)(B)))                                                 01250
((MONOMORPHISM(F1 A B))((A)(B)))                                                01260
((ISOMORPHISM(F1 A B))((A)(B)))                                                 01270
((SUBGROUP A B)((A B))(A B))                                                    01280
((IMAGE A (F1 B C))((A C))(A C))                                                01290
((KERNEL A (F1 B C))((A B))(A B))                                               01300
((RIMAGE A B (F1 C D))((B C)(A D))(B C)(A D))                                   01310
((INVIMAGE A B (F1 C D))((A C)(B D))(A C)(B D))                                 01320
((SUBFGRP A B C D)((A)(B C D))(B C)(B D)(C D))                                  01330
((ABELIAN A B)((A B))(A B))                                                     01340
((CENTER A B)((A B))(A B))                                                      01350
((INTERSECTION A B C)((A B C))(A B)(A C))                                       01360
((NORMAL A B C)((A B C))(A B)(A C)(B C))                                        01370
((NORMALIZER A B C)((A B C))(A C)(B C))                                         01380
((CONJUGATE A A1 B C)((A B C))(A C)(B C))                                       01390
((RCOSET A A1 B C)((A B C))(A C)(B C))                                          01400
((LCOSET A A1 B C)((A B C))(A C)(B C))                                          01410
((SETINV A B C)((A B C))(A C)(B C))                                             01420
((INVPNT A A1 (F1 B C))((A B))(A B))                                            01430
((CENTRAL A B C)((A B C))(A C)(B C))                                            01440
)                                                                               01450
```

```
                        COMPIL        LISP
COMMON((W1 W Z4 N QF1 VV1 VV2 QM5 CX))                                  00010
LOAD((PUT2))                                                            00020
COMMON((QM1 QM2 QM3 QM4 QM6 ZN))                                        00030
COMMON((Z1 Z2 Z3 Q QF2 XX QL1 QL2))                                     00040
COMMON((X Y Z6 FQM2 FQM3 FQM4 CL Z4A Z4B CX1 LON))                      00050
LOAD((SOLVEX))                                                          00060
LOAD((VERIFY))                                                          00070
LOAD((PUTON1))                                                          00080
COMPILE((PUTON1))                                                       00090
LOAD((GTFRM))                                                           00100
COMPILE((GTFRM))                                                        00110
LOAD((HFM))                                                             00120
COMPILE((HFM))                                                          00130
LOAD((GMATCH))                                                          00140
LOAD((NEXT))                                                            00150
LOAD((ROFV))                                                            00160
COMPILE((NOLOOK SPREAD ASTERISKED FIFTH SIXTH))                         00170
COMPILE((ALLSUBSTS ASSOCM PMATCH ROFV CHNGW))                           00180
LOAD((SCNX))                                                            00190
COMPILE((SCNX))                                                         00200
LOAD((INVM))                                                            00210
COMPILE((INVM2 SIFT))                                                   00220
COMMON((SUP))                                                           00230
LOAD((VBLES))                                                           00240
COMPILE((SUPXEC SUPSUB))                                                00250
LOAD((RPTSUB))                                                          00260
LOAD((GFIND))                                                           00270
LOAD((MATCH))                                                           00280
LOAD((ATMS))                                                            00290
COMPILE((ATMS GENVBLE RPLCEF))                                          00300
LOAD((HOMOMF))                                                          00310
COMPILE((HOMOMF ONLY))                                                  00320
LOAD((PUTON2))                                                          00330
LOAD((MODEL))                                                           00340
COMPILE((MODELMAKE CHECKMODEL))                                         00350
COMPILE((MEET SWMEMB NOWWHICH GENSET))                                  00360
LOAD((SLVX))                                                            00370
COMPILE((SLVX))                                                         00380
LOAD((MONOMF))                                                          00390
COMPILE((MONOMF))                                                       00400
LOAD((WLDFN))                                                           00410
COMPILE((COMPOSITION))                                                  00420
EXCISE(*T*)                                                             00430
```

BIBLIOGRAPHY

1) Black, F., 1964. A Deductive Question Answering System, doctoral dissertation, Harvard University, Cambridge, Mass.

2) Crisman, P. A. (ed.), 1965. The Compatible Time-Sharing System - A Programmer's Guide (Second Edition), Cambridge, Mass.: M.I.T. Press.

3) Davis, M., and Putnam, H., 1960. A computing procedure for quantification theory, Journal of the Association for Computing Machinery, July, 7:201-215.

4) Fenichel, R. R., and Moses, J., 1966. A New Version of CTSS LISP, Artificial Intelligence Memo 93, Massachusetts Institute of Technology (Project MAC), Cambridge, Mass.

5) Gelernter, H., 1959. Realization of a geometry theorem-proving machine, in [A], pp. 134-152.

6) Gelernter, H., Hansen, J. R., and Loveland, D. W., 1960. Empirical exploration of the geometry theorem machine, in [A], pp. 153-163.

7) McCarthy, J., 1959. Programs with common sense, Proceedings of the Symposium on Mechanisation of Thought Processes, National Physical Laboratory, Teddington, England, London: H. M. Stationery Office, pp. 75-84.

8) McCarthy, J., et al., 1962. LISP 1.5 Programmer's Manual, Cambridge, Mass.: M.I.T. Press.

9) McCarthy, J., 1963. Situations, Actions, and Causal Laws, Stanford Artificial Intelligence Project Memo 2, Stanford University, Palo Alto, Calif.

10) Minsky, M. L., 1956. Notes on the Geometry Problem, I and II (mimeographed), Artificial Intelligence Project, Dartmouth College, Hanover, Vt.

11) Minsky, M. L., 1961. Steps toward artificial intelligence, in [A], pp. 406-450.

12) Minsky, M. L., 1965. Matter, mind and models, in [B], pp. 45-49.

13) Newell, A., and Ernst, G., 1965. The search for generality, in [B], pp. 17-24.

14) Newell, A., Shaw, J. C., and Simon, H. A., 1957. Empirical explorations of the logic theory machine, in [A], pp. 109-133.

15) Newell, A., Shaw, J. C., and Simon, H. A., 1959. Report on a general problem-solving program, Proceedings of the International Conference on Information Processing, Paris: UNESCO House, pp. 256-264.

16) Polya, G., 1954. How to Solve It, Princeton, N.J.: Princeton.

17) Polya, G., 1954. Mathematics and Plausible Reasoning, Princeton, N.J.: Princeton, 2 vols.

18) Robinson, J. A., 1963. Theorem-proving on the computer, Journal of the Association for Computing Machinery, April, 10:163-174.

19) Robinson, J. A., 1965. A machine-oriented logic based on the resolution principle, Journal of the Association for Computing Machinery, January, 12:23-41.

20) Safier, F., 1963. "The Mikado" as an Advice Taker Problem, Stanford Artificial Intelligence Project Memo 3, Stanford University, Palo Alto, Calif.

21) Samuel, A. L., 1959. Some studies in machine learning using the game of checkers, in [A], pp. 71-105.

22) Scott, W. R., 1964. Group Theory, Englewood Cliffs, N.J.: Prentice-Hall.

23) Slagle, J., 1961. A heuristic program that solves symbolic integration problems in freshman calculus, in [A], pp. 191-203.

24) Slagle, J., 1965. Experiments with a deductive question-answering program, Communications of the Association for Computing Machinery, December, 8:792-798.

25) Teitelman, W., 1966. PILOT: A Step Toward Man-Computer Symbiosis, doctoral dissertation, Massachusetts Institute of Technology, Cambridge, Mass.

26) Wang, H., 1960. Toward mechanical mathematics, IBM Journal of Research and Development, January, 4:2-22.

27) Wang, H., 1960. Proving theorems by pattern recognition - I, Communications of the Association for Computing Machinery, April, 3:220-234.

28) Wang, H., 1961. Proving theorems by pattern recognition - II, Bell Systems Technical Journal, January, 40:1-42.

29) Wos, L., Robinson, G. A., and Carson, D. F., 1965. Efficiency and completeness of the set of support strategy in theorem proving, Journal of the Association for Computing Machinery, October 12:536-541.

[A] Feigenbaum, E. A., and Feldman, J. (eds.), 1963. Computers and Thought, New York, N.Y.: McGraw-Hill.

[B] Kalenich, W. A. (ed.), 1965. Information Processing 1965 (Proceedings of IFIP Congress 65), International Federation for Information Processing, New York City, Washington D.C.: Spartan.

INDEX

(Underlined references indicate definitions)

advice-taker, 7, 101 ff, 127 ff
argument (of object or statement), 16, 17
backtracking, 10, 68, 73
"built-in" axioms, 33, 56 ff
"change of variables", 50
connective, 14
constant, 13, 20
    logical constant, 14
    syntactic constant, 15
construction, 77, 86, 88, 113
definitions, 15, 19, 24, 25 ff
detachment, 41, 58, 107, 109, 112
EQUAL, 14, 32, 38, 44, 87
EQUAL2, 14
existential generalization, 47
existential specification, 45, 47
EXISTS, 15
"explore consequences", 25 ff, 34, 106
FEQUAL, 14, 50, 80
GENFCN, 50, 81
head (of tree), 21
HOMOMF, 26, 50, 105
IMPLIES2, 14
ISOLVE, 50 ff, 64, 81 ff, 105
isomorphism, (see ISOLVE)
lemma, 34, 36, 37, 41 ff, 72 ff, 91
LISP, 13
logical class, 21, 34, 59, 107
logical constant, 14
matching, 55, 57, 124
mathematical induction, 94, 96 ff
MODEL heuristic, 44, 45, 46, 61 ff, 81, 90, 110
necessary condition, 27, (also see definitions)
object, 15
    compound object, 16
Project MAC, 13, 23, 28
progress, 10, 29, 34, 41, 58 ff, 112 ff
    heuristic A, 58 ff
    heuristic B, 59, 87, 107, 112
property list, 19
    of a condition or definition, 19
    of a table I entry, 21, 30
    of a table II entry, 21 ff, 30
PNT, 23
PUTON1, 45
PUTON2, 46
quantifier, 15, 20, 48
reduction, 21
related substitution, 33, 34, 39, 52, 73, 74, 75
scanning, 33 ff
SCANW, 26, 34 ff, 49, 50, 54
SCNX, 37, 49
semantic model, 61 ff, 126 ff
SLVX, 46, 49, 91
SOLVEX, 26, 48 ff, 90
statement, 17
status, 23
subordinate tree, 23, 37, 42 ff, 61 ff, 72 ff
subscript, 94 ff
sufficient condition, 15, 19, 24, 25 ff

symbol, 13
syntactic constant, 15
table I, 19, 21, 25 ff
table II, 19, 21 ff, 25 ff
table entry, 19
term, 15
    asterisked term, 15, 16, 18, 53, 91, 92n
    non-asterisked term, 15, 17, 18, 53, 91, 92n
time, 64, 81, 100, 102
tree, 21 ff, 47
type, 13
universal specification, 28
variable, 13, 20
verification, 47 ff
working backward, 8, 21, 25, 28, 122, 124
working forward, 42, 53, 89

BIOGRAPHY OF THE AUTHOR

Lewis Mark Norton was born in Derby, Connecticut on August 18, 1940. He attended public schools in Shelton, Connecticut and was graduated valedictorian of his high school class in 1958. He attended M.I.T. as an undergraduate with the aid of a National Merit Scholarship, receiving his S.B. in Mathematics in June, 1962. He then enrolled at M.I.T. as a graduate student and held a National Science Foundation Cooperative Graduate Fellowship for the academic year 1962-63. Since 1963 he has been a Research Assistant associated with Project MAC, and expects to receive his Ph.D. in Mathematics in September, 1966.

The author has been employed by Charles W. Adams Associates, Incorporated, Bedford, Massachusetts, Northern Research and Engineering Corporation, Cambridge, Massachusetts, and Sikorsky Aircraft, Stratford, Connecticut, and has accepted a position with the MITRE Corporation, Bedford, Massachusetts, effective September, 1966. Since 1962 he has been a member of the Association for Computing Machinery, the American Mathematical Society, and the Mathematical Association of America, and an associate member of the Society of the Sigma Xi.