UNIVERSITY OF CAPE TOWN

DEPARTMENT OF COMPUTER SCIENCE

# A PROBLEM SOLVING SYSTEM EMPLOYING A FORMAL APPROACH TO MEANS/ENDS ANALYSIS

by

GAVIN ROSS FINNIE

A Thesis
Prepared under the Supervision of
Prof. K.J. McGregor in Fulfilment of the Requirements for the
Degree of Master of Science in Computer Science

CAPE TOWN

April, 1976

# LIST OF CONTENTS

---oo●oo---

# ABSTRACT

The thesis describes the theory and design of a general problem solving system. The system uses a single general heuristic based on a formal definition of differences within the framework of means/ends analysis and employs tree search during problem solution. A comparison is made with two other systems using means/ends analysis. The conditions under which the system is capable of solving problems are investigated and the efficiency of the system is considered.

The system has solved a variety of problems of varying complexity and the difference heuristic appears comparatively accurate for goal-directed search within certain limits.

# 1. INTRODUCTION

## 1.1 Introduction

One of the often stated basic goals of Artificial Intelligence research has been the construction of machines which perform tasks requiring some form of 'intelligence' {9, 21, 28}. Since a good deal of natural intelligence is involved in solving everyday problems, the study of the concepts and techniques of problem solution has long been an active area of research in computer science.

Clarity is required as to the question of the scope of the study of problem solving. It has been observed by Ernst and Newell {9} that from the user's point of view a computer is a general problem solver and that any working set of programs is in fact the solution of some problem. However problem solving at this level is usually not considered and most work in the field has been more concerned with the discovery of general rules and methods involved in the solution of problems rather than with the attainment of a solution for any particular problem.

The thesis describes the design and implementation of the problem-solving system SDPS (Syntactic Deductive Problem Solver). SDPS is intended as a general purpose problem solver in that it can deal with a wide variety of problems within a single type of problem formulation. It uses a single general heuristic technique for goal-directed tree search. The system was developed largely to investigate the heuristic power of the method and to consider the effective generality of the heuristic technique. The SDPS system is written in a version of ALGOL compatible with the NUALGOL compiler for the Univac 1106. Algol was selected mainly for reasons of efficiency of execution as this broadens the universe of problems which may be considered. A listing of the system is given in Appendix B.

SDPS uses the general concept of means/ends analysis for goal-directed search. Means/ends analysis has featured in the design of a number of problem solving systems, e.g. GPS {9}, FDS {22, 23} and STRIPS {12}. Means/ends analysis consists essentially of establishing some measure of differences between a given problem object and a goal object and of using these differences to direct the search for a solution which consists of a sequence of object transformations until the goal object is attained. The basic model of a problem used by such systems is given in section 1.4. Chapter Two contains a brief outline of the GPS and FDS systems and considers their relationship to SDPS.

The differences used by SDPS are established by the use of a specific object representation and a formal definition of the differences which may occur between two objects in terms of their constituent elements at particular positions in the representation. Chapter Three is devoted to a summary of the SDPS system design. The object representations are described and the formal concept of differences defined. The use of these differences for the selection of operators which transform an object towards the goal representation is explained. SDPS employs a general disjunctive tree search and the use of the heuristic for ordering nodes is discussed. Tree search enables the use of some standard measures of heuristic power, namely penetrance {6} and effective branching factor {21}.

The effective generality of the system is essentially a consideration of the type of problem SDPS is capable of solving. Chapter Four considers this question rather formally by the use of a model of a problem and the establishment of conditions under which SDPS will obtain the solution to a problem. The algorithm used by SDPS is also given here.

The last chapter defines the measures of efficiency used by SDPS, and gives some examples of the type of problem solved by the system.

The rest of the introductory chapter considers the two major approaches to problem solving systems and the conflicting aims of generality and efficiency. It also defines the basic concepts of problems and heuristic search used by systems like SDPS.

## 1.2 Approaches to Problem Solving

There have been two major lines of attack in computer studies of problem solving. The first has been to develop problem solving systems which serve as a model of cognitive processes for use as an aid to understanding natural (human) intelligence. This is primarily an approach from the field of psychology. An example is the work of Newell and Simon {20} in which a theory is constructed which considers a person as an information processing system (IPS). A model of an IPS is developed and applied to specific task environments, and an attempt is made to ally these results to those of humans involved in similar environments. The General Problem Solver (GPS) of Newell, Shaw and Simon {9} was originally developed for studying natural intelligence.

The second approach is that of building systems which will solve problems irrespective of whether they use human methods or not, i.e. the 'intelligence' they exhibit need have no relation to natural intelligence. One example here is theorem proving programs employing the resolution principle {28}.

The line taken in the SDPS system falls somewhere between these two extremes. Although a descendant of GPS employing the same technique of means/ends analysis as a heuristic, the method of obtaining the heuristic information is probably closer to the second approach than to the first.

## 1.3 Efficiency and Generality

Another area in which conflicting approaches have been made to problem solving is on the question of the degree of generality or expertness of the system. Questions of generality concern the breadth of the universe of problems a problem solver is prepared to work in and the generality is achieved by the use of universal methods and universal problem representations. The expertness of a problem solver is measured by the quality of the answers achieved.

In general it may be said that the more general a problem solver the less efficient it is. To quote Feigenbaum {11}:

> 'A view of existing problem solving programs would suggest, as common sense would also, that there is a kind of "law of nature" operating that relates problem solving generality (breadth of applicability) inversely to power (solution successes, efficiency, etc.) and power directly to specificity (task specific information).'

As GPS was originally designed to model natural intelligence, little attention was paid to the quality of problem solving. The SDPS system uses the same universal concepts as GPS and as a result suffers to some extent from the lack of problem specific heuristics.

## 1.4 Heuristic Search in Problem Solving

The following formulation of a problem has been described previously {2}, {8} and has been called the problem solving problem. A task environment always contains a set S of problem situations and a set F of operators which may be applied to elements of S. Given an initial situation $s \in S$ and a set of desired situations $\omega \subseteq S$, a solution to the transformation problem is then a sequence of operators $f_1, f_2, .., f_n$ such that $f_i \in F$ for $i = 1,2,..,n$ and

$$f_n(f_{n-1}(\ldots f_1(s)\ldots)) \in \omega$$

Most problem solvers attack this problem by searching the tree of all possible operator applications. The operators are in effect partial functions since not every operator is applicable to every problem situation. Heuristic search is used if the order in which the nodes are selected is determined by the heuristic properties of the nodes themselves. The heuristics may be any features of the task environment which suggest the potential location of the goal. Heuristic search is obviously essential for any non-trivial problem as the complete problem tree may be of infinite size.

# 2. MEANS/ENDS ANALYSIS IN PROBLEM SOLVING

## 2.1 Means/ends Analysis

Means/ends analysis is a general heuristic search technique employed to order the selection of operators to be applied to problem states {9, 28}. An operator is selected as a function of the differences between the given state and the required state - selection being based on the probability that application of the operator will remove at least one difference between the states. Differences may be defined in a number of ways, e.g. they may be a list of features which occur in one state but not in the other, or they may be a partial list of reasons why the given state does not satisfy a test for the goal state, etc.

Problem solvers based on means/ends analysis usually employ a recursive problem reduction approach. If an operator is judged as likely to remove a difference and the operator is not immediately applicable to the current state, a subproblem is set up to transform this state into one in which the operator is applicable.

Nilsson {21} has introduced the concept of 'key operators', i.e. operators which must be applied at some stage in the solution sequence. Differences may be used to identify such potential key operators. The original problem then reduces to the subproblem of transforming the initial state to a state in which the key operator is applicable, and the subproblem of transformation from this state to the goal state. The subproblems may of course themselves be reduced to a set of subproblems.

Some of the importance of the means/ends approach lies in the fact that it appears to be a general technique employed by most human problem solvers in certain task environments {17}.

## 2.2 The General Problem Solver

The GPS was originally envisaged in 1957 and existed in a number of forms until 1969. It is a very general multipurpose problem solving program employing the heuristic technique of means/ends analysis. The final version has been very completely documented in {9}. The following is a brief summary of certain features relevant for comparison to the SDPS system.

A problem as specified for GPS consists of:

(1) An initial object;

(2) A set of desired objects;

(3) A set of operators.

Objects are represented as a general tree structure, each node having an arbitrary number of branches. Each node may also have a local description consisting of a number of attribute-value pairs.

Two types of operator occur in GPS. The operators transform objects into new objects. Schema operators are represented as a pair of objects containing variables: the first object giving the form of the input, and the second giving the form of the output. Move-operators are somewhat more flexible. These consist of a set of constraints and a set of transformations: the transformations indicate how the input is to be modified and the constraints specify the conditions under which the operator may be applied.

In addition to the problem formulation, it is necessary to provide, among other things, the following:

(1) A set of differences;

(2) A table-of-connections;

(3) A difference ordering.

The table-of-connections provides an explicit user-defined link between the differences and the operators relevant to removing them. Differences in GPS are user-specified and the differences detected during problem solving consist, of a difference type, difference value and the position of the node where the difference occurred. Operators are selected by retrieving from the table-of-connections those operators linked to difference type. The differences are ordered in terms of degree of difficulty.

GPS uses the standard recursive approach to tree search outlined in 2.1, but in fact employs four general types of goal. These are:

(a) Transform object A into object B;

(b) Reduce difference D on object A;

(c) Apply operator Q to object A;

(d) Select the elements of set S which best fulfil criterion C.

The solution procedure is roughly as follows: If a difference D is detected between objects A and B during any attempt to achieve a goal of type (a), then a subgoal of type (b) is set up. If the table-of-connections indicates that an operator Q is applicable to reducing D it is applied if possible otherwise a subgoal of type (c) is set up to make it applicable. Goals of type (d) were introduced in later versions of GPS to handle situations in which it is necessary to select elements of some set of objects on the basis of their similarity to a required object structure.

The type of search is essentially depth first - GPS works on a goal for as long as it seems desirable. Sandewall {25} has called this the labyrinthine approach. GPS requires differences to get easier and easier as problem solving progresses. An operator is rejected if it leads to a difference more difficult than the difference for which the operator was selected.

GPS has solved a wide variety of problems {9} but is on average a very slow performer. However it can work on problems requiring both inductive and deductive reasoning.

### 2.2.1 Some Limitations of GPS

The slow speed of GPS limits the variety and complexity of problems it can be applied to.

Labyrinthine search tends to limit the attention of GPS to one particular area of the goal tree for considerable periods of time. The program requires a more global view of the entire task environment and requires the ability to select goals globally rather than locally.

The problem solving actions and the efficiency of GPS are strongly related to the particular problem representation selected by the user.

## 2.3 The Fortran Deductive System

The FDS system {22, 23} was developed in the late 1960's. It is to some extent a descendant of GPS, employing the same heuristic technique of means/ends analysis.

A problem is specified to FDS as:

(1) An initial object;

(2) A desired object;

(3) A set of operators.

All objects are represented as prefix polish strings. Differences between objects are determined by testing corresponding elements in the strings. In contrast to GPS there is no explicit linking of operators and differences, and no definition of the differences is supplied by the user. The system itself sets up tables to detect whether an operator is relevant to reducing a difference.

The operators are specified in the form of compiler-like productions. Similar to the GPS schema-operator, they consist of a pair of objects: the first object specifying the input and the second the output. There is no FDS analogue of the GPS move-operator.

FDS differs from most problem solvers in that it does not employ tree search. Instead a top-down depth first approach is used.

The procedure is roughly as follows.

The top level consists of the initial object s, the desired goal g and an ordered set of operators relevant to removing differences between the strings. The ordering of the operators is based on the probability that the operator will remove a difference.

The first operator is selected from the list and matched with string s. If it can be applied, a new string s' results. The level is increased by one and the initial and goal string at this level are s' and g respectively. If an operator is not applicable a subgoal g' is set up, the level is increased by one, and the initial and goal string are s and g' respectively. A new ordered set of operators is generated for this level.

The procedure continues in this way. If a subgoal is solved the operator which gave rise to it is applied and search continued. As each new (s,g) pair is generated a goal test is applied.

If the depth bound is exceeded without a solution being obtained, the level is decreased by one and the procedure restarted. If all the operators at a level are exhausted search is restarted at the next higher level.

Search continues until either a solution is obtained, the allotted time is exhausted or all operators have been attempted without success.

### 2.3.1 Some Limitations of FDS

The major drawback of the FDS system lies in the top-down approach. Although it prevents the explosive growth of nodes which may arise in standard tree-search procedures, efficient search requires a highly selective ordering of the operators to be applied at each level. If an incorrect operator is selected at a fairly high level above the depth bound, the search below that point will effectively be blind in that all operators below that level must be exhausted before control returns to the level. This type of search gives little idea of the heuristic power of the methods used.

By overwriting paths which may already have occurred at a lower level, FDS tends to repeat steps until a sufficiently high level is reached for a complete solution sequence to be obtained. This type of repetition is far simpler to isolate in tree search and again detracts from the efficiency of the system.

The only criterion of efficiency used in FDS is that of time to solution. This makes it difficult to draw comparisons with other problem solving systems as the time taken is to a large extent dependent on the language used, the machine the problem solver is implemented on, etc. A measure of efficiency such as penetrance {6} in tree search would enable a better test of the formal type of means/ends analysis used in FDS.

The lack of an operator similar to the move-operator of GPS makes the formulation of certain type of problem extremely awkward. However this type of operator would be very difficult to incorporate in the FDS structure.

## 2.4 The SDPS system

The problem solver under consideration was originally developed along the lines of the FDS system. As a result the formal concepts of operators and differences are similar to those used in FDS.

When the problems inherent in the top-down approach to search were discovered by practical observation, it was decided to adopt the more conventional method of tree search. However the approach taken is not that of the GPS labyrinthine search but is more similar to the backing-up techniques of MULTIPLE {27, 28}. When an operator has been applied or a new subgoal set up, the new node is evaluated and this value backed up through the tree. Each node in the tree has associated with it the name and value of its best successor. It is then a fairly simple procedure to determine the potentially best node in the entire tree and this node is selected for expansion. Sandewall {25} refers to this as the best-bud method of tree search and the intention of using it is to get an overall view of the partial state of solution of the problem. The method differs from that of MULTIPLE in that only one successor of a node is generated at a time whereas MULTIPLE expands all immediate successors before evaluating the nodes.

SDPS employs only one type of goal as opposed to the four used by GPS. This goal is the equivalent of GPS goal type (c), i.e. apply operator Q to object A. GPS goal types (a) and (b) are implicit in the SDPS design and there is no SDPS analogue of goal type (d).

The SDPS system is thus a general problem solving program employing heuristic search techniques based on a formal concept of means/ends analysis. It defines its own differences and table-of-connections and employs a general technique of tree search to discover a solution sequence of operators.

# 3. THE SDPS SYSTEM

## 3.1 The Task Environment

The system works within the framework of the standard heuristic search problem paradigm. A problem specification consists essentially of a triple (s, F, t) where s is an initial (given) object, t is a desired object and F a set of operators. The operators transform object states to new object states in the state space. A problem is considered solved when a solution sequence is obtained, a solution sequence being a sequence of operator transformations

$$f_n(f_{n-1}(\ldots.f_1(s)\ldots.)) = h$$

where h is equivalent to the goal object t.

No attempt is made to optimize the solution sequence in the sense of finding the shortest path from the initial state to the goal state.

## 3.2 The Representation of Objects

The set of symbols used to represent objects consists of a finite set of constant symbols C and a countably infinite set of variables V. These form the alphabet of the problem space.

The constant symbols are programmer-defined and are specific to the problem under consideration. They provide the context of the problem.

Formally, the set C consists of the union of all sets

$C_i$ where $C_i$ is the set of all constant symbols of degree i. The sets are non-intersecting, i.e. no constant symbol may have varying degree.

e.g. in the context of propositional calculus, the set $C_0 = \{P, Q, R\}$ where P, Q, R are propositions of degree 0, $C_1 = \{\sim\}$, a unary operator, and $C_2$ the set of binary operators $\{\wedge, \Rightarrow, \vee\}$.

The variables V are not problem specific - they are considered as free variables and are represented as $V_i$, $i > o$. e.g. $V_1 \wedge V_2$.

The use of constant classes is a convenient method of grouping similar constant symbols for various types of problem, e.g. the use of classes of similar operators in group theory.

The classes form a cover D for the set of constants where $D = \bigcup_i D_i$ $(i = 1,...,m)$. All the constants in $D_i$ are of the same degree for all i and every constant symbol is in at least one $D_i$.

All constant symbols are held in a symbol table giving their degree, class, etc.

Objects in SDPS are represented conceptually by tree structures. The constant symbols of degree greater than zero form the non-terminal nodes, variables and constants of degree zero form the terminal nodes. Formally an object may be defined as a well formed structure as follows:

(1) A variable or constant of degree zero is a well formed structure.

(2) A node of degree n with n ordered successor well formed

structures is a well formed structure.

The ordering concept is necessary to allow comparison between structures.

e.g. in elementary algebra, the expression ((-A) + B * (C-D))/E could be represented by the tree in Fig. 3.1.

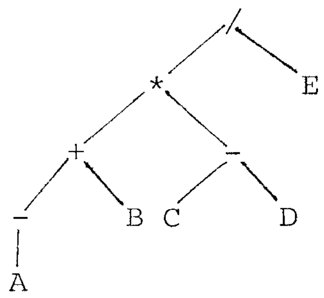



Figure 3.1

Note that the first minus sign is unary. The trees as defined are n-ary.

Two object structures may now be compared in terms of the relative positions of their substructures. This requires some method of numbering or ordering the nodes to allow direct references to any subsection of the tree. The nodes of the structure are numbered in the order in which they would be visited by some fixed technique of traversing the tree - in SDPS pre-order traversal is used and any reference to traversing an object tree will mean pre-order traversing. Pre-order traversal means that the root node is the first visited and is assigned the positional value of one. Any other method of traversing or numbering could be used provided consistency is maintained.

Pre-order traversal for binary trees is defined recursively by Knuth {14} as:

(1) Visit the node;

(2) Traverse the left subtree;

(3) Traverse the right subtree.

Although the object structures are in fact n-ary trees, any n-ary tree may be simply transformed to a binary tree {14}. The transformation is achieved by linking together the sons of each node and removing the vertical links except between a father and his first son.

The system does not, as yet, consider an object as a forest, where a forest is defined as an ordered set of 0 or more trees. This is possibly a more flexible approach than the above, as an object structure could be considered as a set of attributes.

In practice it is found that virtually all object structures are already binary trees, as the operators in most theories considered are either unary or binary.

As a basis for comparison between objects and to facilitate the discovery of differences between them the following terms must be defined.

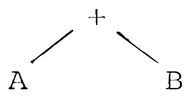
The size of the tree N(s) is the number of nodes in tree s.

The symbol $s_i$ is the value of the i'th node (as determined by the traversing).

S(i) is defined as the subtree rooted at node i.

The direct successors of any node are ordered in terms of first son, second son, etc. The ordering is from left to right and any reference to the i'th direct successor of node j is defined by the relationship in which the sons stand to the parent node.

e.g.




A is considered the first son, B the second.

The tree structures used are usefully flexible as virtually any problem object can be defined in terms of them.

3.3 The Storage and Retrieval of Objects

Objects in SDPS are stored by filing them in a binary tree structure similar in concept to the canonical tree of GPS.

To facilitate comparison between problem structures in the goal tree each object is given a unique name when it is first generated. The objects are filed in node number order.

The nodes in the discrimination tree contain 5 items, packed for storage efficiency:

(1) The value of the node;

(2) The name of the node;

(3),(4),(5)

The left branch, right branch, and the parent of the node.

The boolean procedure NAMELT is used to file the strings and to determine whether the particular string is already in existence. Filing is done by comparison between the value of the node and the value at the current position in the string. If a match is obtained the right branch is taken and the string pointer incremented; if there is no match the left branch is taken. If the end of the string is reached, the node is tested to determine whether the string has been named or not. If at any stage of the procedure the right branch is empty, the rest of the string is filed to the right of the node. If the left branch is empty, the first value is filed to the left of the node and the rest of the string filled in to the right of the new node.

e.g. Given structures with ordered nodes to be filed

- + A B C
- B C
+ A B
- + A B D

the tree would be

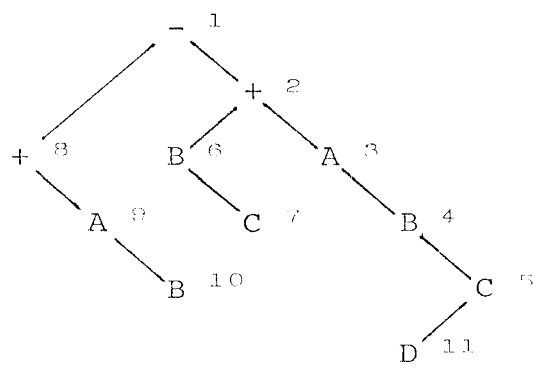



Figure 3.2

The numbers indicate the order of filing.

The filing procedure allows for fairly quick identification and storage of objects. It is well suited to the notation used as a large number of the objects generated during solution of a particular problem have the same initial sequence of symbols, leading to the saving of a quite considerable amount of storage.

The name of the structure is the number of the last node in the string, e.g. in the above tree the object - + A B D has name 11.

The structure name is used to retrieve strings from the tree. The object is obtained by backing up from the named node to the top node, returning in order the values of those nodes reached by a right branch from the parent.

Once a string has been retrieved it is transformed (procedure POSMAP) into a tree-like structure by providing forward and backward links between substructures. This mapping facilitates manipulation of the objects during the detection of differences and the selection of operators.

## 3.4 THE COMPARISON OF OBJECTS

To facilitate the comparison of objects it is necessary to consider the following concepts.

Roughly speaking two structures are equivalent if they have the same shape and each node contains the same information, i.e. they have the same interpretation within the problem environment. Formally two objects s and t are equivalent if $N(s) = N(t)$ and for every ordered node i in the structures either

(i) $s_i = t_i = V_j$ for some j, i.e. both nodes equal the same variable, or

(ii) $s_i, t_i \in D_k$ for some k.

Substitution for any of the terminal nodes $V_i$ is allowed, provided this substitution is consistent throughout the structure. A substitution function sub $(V_i, u, s)$ is defined as the object structure which results from replacing each occurrence of the variable $V_i$ in the structure s by the well-formed structure u.

A structure may be a substitution instance or specification of another structure, written s S t. Formally s S t if there exists a substitution sequence $(\text{sub } (V_{ij}, U_j, s), j = 1,\ldots,n)$ such that s and $\text{sub } (V_{in}, U_n\ (\text{sub } (V_{in-1}, U_{n-1}\ (\ldots(\text{sub } (V_{i1}, U_1, t))))))$ are equivalent.

The structures s, t in Figure 3.3 are s S t.

e.g.

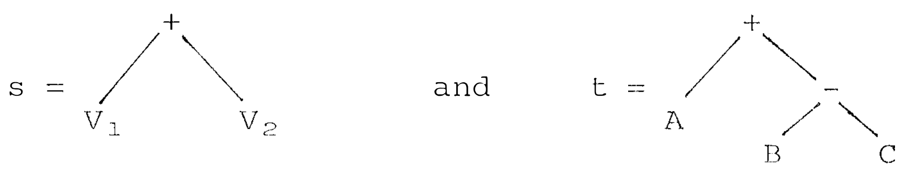



Figure 3.3

The relationship of correspondence is used to compare elements within structures. It may be defined recursively as follows. Given two object structures s and t

(i) $s_1$ C $t_1$, i.e. the root nodes correspond,

(ii) $s_i$ C $t_j$ if there exist nodes $\ell$, m such that

(a) $s_\ell$ C $t_m$

(b) $s_\ell$ S $t_m$

(c) s(i) and t(j) are the n'th ordered sons of nodes $s_\ell$, $t_m$ respectively.

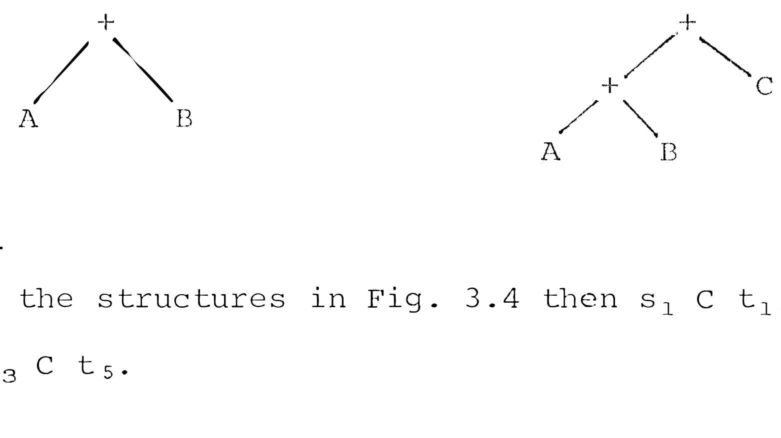



Figure 3.4

e.g. given the structures in Fig. 3.4 then $s_1$ C $t_1$, $s_2$ C $t_2$ and $s_3$ C $t_5$.

## 3.5 A FORMAL CONCEPT OF DIFFERENCES

Differences between structures are selected by the syntactic concept of elements which correspond to each other. Differences occur when two corresponding elements are not specifications of each other. If the element in the second object is a variable, the first element is substituted for it throughout the object and differences are again taken.

The advantage of this definition lies in its generality - it is in no way dependent on the particular task under consideration.

A difference between two objects s and t is an ordered pair (t', k) where t' = some $t_j$ and $t_j$ C $s_k$.

The difference set between two structures s and t is defined as

(1) The set of pairs (t', k) such that t' = some $t_j$ and

 (a) $t_j$ C $s_k$

 (b) $t_j$ is not a variable

 (c) $t_j$ is not a specification of $s_k$.

(2) The set of pairs (t', k) such that t' = $t_j$ and there exist $\ell$, m with the properties

 (a) $s_m$ C $t_\ell$

 (b) $t_\ell$ is variable

 (c) (t', k) belongs to the difference set between s and sub ($t_\ell$, s(k), t).

e.g. given the two objects in Fig. 3.5,


Figure 3.5

the differences would be (a, 3) by (1) above, and (c, 5), (b, 4) by (2) above.

This definition of differences is rather limited in scope and in certain circumstances provides not much knowledge about the problem under consideration.

e.g. between structures in Fig. 3.6, the only difference detected is (-, 1).


Figure 3.6

3.6 THE REPRESENTATION OF OPERATORS

Operators in SDPS are held in the same form as schema operators in GPS. An operator consists of a pair of objects, written I: = O, in which the first (left hand) object gives the form of the input and the second (right hand) object gives the form of the output. The operator objects usually contain variables

e.g. $f_1$: $V_1 + V_2 - V_2 := V_1$

Operators are applied to an object by matching the input of the operator either to the current structure or to some substructure within the object. If the structure is not a substitution instance of the operator input, the operator cannot be applied. If it is a substitution instance, the particular set of substitutions required are isolated and are used to replace the same variables in the output object.

The set of operators is called F and individual operators are $f_i \in F$, $i = 1,\ldots,n$. Operators may be applied at any node in the object. The notation $f_{ij}$ will mean that operator $f_i$ is to be applied to the structure rooted at node j.

e.g. given rule $f_1$ above to be applied to the object is Fig. 3.7(a) at the top node, the result of $f_{11}(s)$ is Fig. 3.7(b).

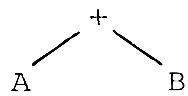



Figure 3.7(b)

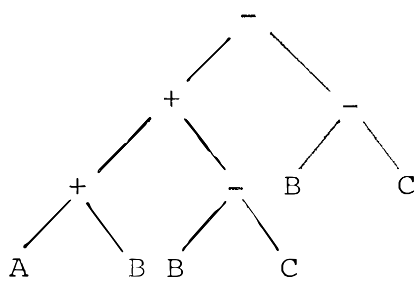



Figure 3.7(a)

The substitutions required to make the object a specification of the operator input are ($V_1$, + AB), ($V_2$, - BC).

In using the concept of differences to direct the search for a solution it is necessary to have some technique of linking differences with those operators likely to remove them. In GPS these links are defined explicitly by the table-of-connections.

To this end it is necessary to have some efficient method of assessing the effect of applying an operator at any node. Even if the operator cannot be applied immediately, there must be some technique of determining the possible effects if it could be applied at a later stage in the solution process. Before initiating the search for a solution the operators are analyzed by means of a rough matching technique between the input and output structures of each operator.

Application of an operator will tend to modify the 'shape' of the object tree as well as changing the values of the nodes. The analysis of the effect of changes is therefore done in terms of the effective position within the well-formed structures, i.e. at those points at which the shape is similar.

e.g. operator $(V_1 + V_2) - V_3 := V_1 - (V_3 - V_2)$ represented in Fig. 3.8

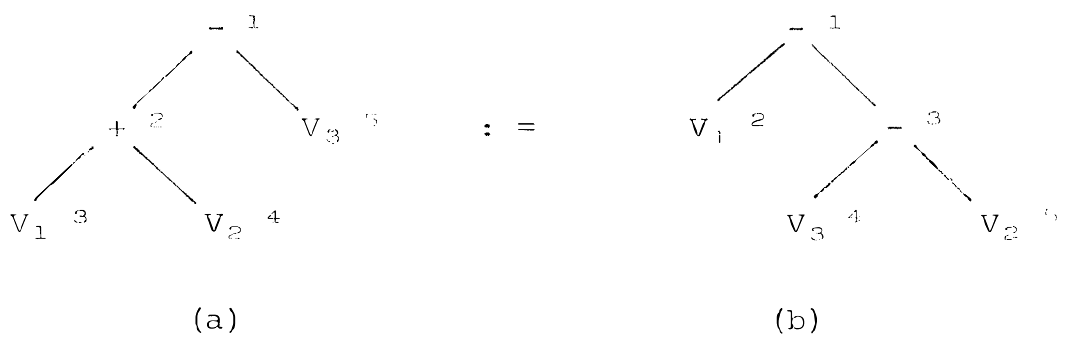



Figure 3.8

The shape of the object has been altered and the effects of the change would be noted in the right hand structure only at those points at which the two structures roughly match, i.e. at node 1, 2 & 3 in (b). Node 1 is unchanged, node 2 has become $V_1$ and node 3 is now a minus sign. The other nodes are effectively ignored.

As differences are defined in terms of elements which correspond to each other it would appear logical to analyze the operators only i.t.o. the differences which arise between the input and output objects - the operators are then capable of removing these differences. This approach was initially attempted and found to be somewhat too restrictive. As a result the concept of comparing only those elements which correspond to each other is not used, i.e. it is not necessary for matched nodes to have parents which are specifications of each other in order to determine the effects of modification.

e.g. rule $V_1 + (V_2 - V_3) := (V_1 + V_2) - V_3$.

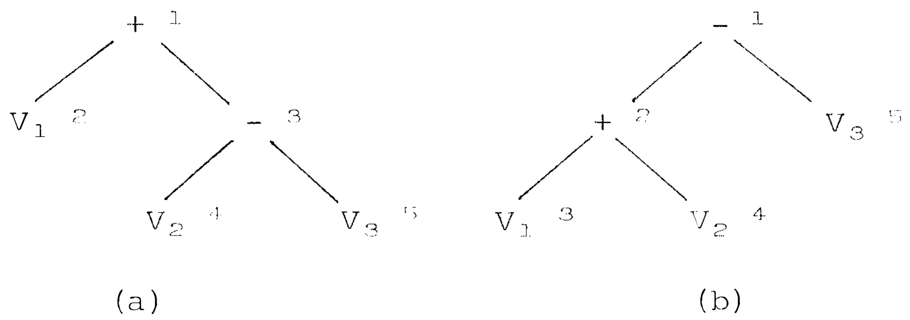



Figure 3.9

The only difference which would be detected is at the top node (-, 1) as any lower nodes would not correspond to each other in terms of the definition. However the analysis is taken a step deeper to include nodes 2 and 5 in (b). This has the effect of providing a deeper knowledge of the operator effect.

When an operator is applied to a structure, two types of symbol may be distinguished in the output object. Firstly there are those symbols which are constants in the r.h.s. of the operator and which remain invariant for any application of the rule. Secondly there are those variable symbols whose values in the output object are dependent on the root. The value recorded here is however a pointer to the second table which records the position(s) of the variable in the input object by showing the relation of the variable to the root node of the input. Again if a variable is in the same position in both the input and output its value is not recorded. The second table may be used to quickly find the substitution value of any variable by applying the same links to the current object.

During analysis a value is associated with each operator as a measure of its complexity. This value is used as a parameter in evaluating the 'worth' of any operator in removing some set of differences. The current tendency is to attempt to use the simpler operators first, as 'more' is known about the effects of an operator application and usually less effort is required to make an operator applicable. The complexity is determined by such factors as the size of the input and output objects (smaller structures being favoured), the difference in size and general shape, the number of positions at which the values are altered, etc.

## 3.7 THE SELECTION OF OPERATORS

The purpose of applying any operator is obviously to reduce the differences between the current object and the goal object. The operators selected must be ordered in terms of their potential usefulness. Similarly to GPS the aim is to select operators which make the problem easier and easier. However whereas GPS will abandon completely a line of approach which is considered to be getting more difficult, such operators in SDPS are not rejected but they receive a low estimate of potential worth. As difficulty of problems can only be measured by the number and type of differences which occur, the aim is to select operators which remove more differences than they introduce.

To select operators a look-ahead procedure, similar in concept to Sandewall's use of images {24}, is carried out. The differences selected by the method of section 3.5 are called zero-level differences. An operator will remove a difference (t', k) if the value at node k is transformed by the operator to be a specification of t'. To achieve this the operator must be applied to some structure containing node k.

Each difference (t', k) is selected in turn and the following procedure applied for each operator $f_i$, $i = 1,...,n$. The structure at node k is isolated and the first entry for the operator in the first table above is inspected. If it is a specification of t' the operator $f_{ik}$ is included as a zero-level operator.

It is then necessary to consider those structures containing node k. $\ell$ is set initially to the parent node of k and the matching procedure applied to the structure at node $\ell$. $\ell$ is then reset to be its own parent node and so on. The cycle of backing up and matching is continued until the root node of the object structure has been dealt with.

In dealing with each structure containing k, the first table is examined to determine whether there is an element loosely corresponding to k, or to some substructure containing k.

If such an entry exists and is a fixed constant which is a specification of t', the operator $f_{i\ell}$ is included in the set of zero-level operators.

If the entry is that of a variable, the second table is used to identify the required substitution in s. If there is an element, say $s_m$, in this substitution structure which matches $s_k$ and is a specification of t', then $f_{i\ell}$ is included as a zero-level operator.

If the element $s_m$ is not a specification of t' the following situation arises. If $s_m$ could be transformed to a specification of t' then the operator $f_{i\ell}$ under consideration could be used to remove the current difference. A new difference (t',m) is thus introduced with the hope that if this difference could be removed, application of the current operator would remove the current difference. The difference (t', m) is added to the set of first-level differences.

When all the zero-level differences have been dealt with the set of first-level differences is handled in exactly the same way. Any operators which remove these differences are placed in the set of first-level operators. Again the examination of these differences may lead to the discovery of second-order differences, and so on.

This 'look-ahead' for potential operators is halted either when a pre-determined level of differences is reached or when the n'th level of differences is empty. No operators or differences are added to a set if they already exist in this set or a lower set.

The selection of operators is based only on the 'rough matching' concept embodied in the tables. There is no test as to whether the structure the operator is to be applied to is a specification of the operator input.

## 3.8 ORDERING OF OPERATORS

Operators must be ordered in terms of their potential ability to remove differences. The node under consideration in the goal tree then retains the ordered list of operators relevant to its own differences.

The factors taken into account in evaluating the worth of an operator include the following:

(1) The various levels at which the operator was generated i.e. the level of difference the operator would remove. If an operator can remove a zero-level difference its value is obviously greater than one which could remove, say, a fourth-level difference.

(2) The number of differences which generated the operator. An operator which can remove a number of differences is of greater value than one removing only one difference.

(3) The complexity of the operator. Simpler operators tend to get preference as there is usually less work involved in making the operator applicable and more is known about the effects of the operator.

(4) Whether an operator contracts or extends the object in relation to whether the current object must be contracted or extended to attain the goal object. The tendency is to modify structures towards the required size.

(5) The potential amount of work required to make the operator

applicable. This is measured by comparing the operator input to the structure and making a quick estimate of the differences. Operators which can be applied immediately have higher value than those which require the setting up of subgoals.

(6) A small factor which relates the size of the object substructure to the size of the operator input structure.

Each of the factors has a bias attached to it which can be varied by the user to increase or decrease the effect of any factor. It is found that in different task environments some factors tend to be more effective than others.

## 3.9 THE STRUCTURE OF THE PROBLEM SOLVING TREE

The problem solving tree is a disjunctive goal tree generated during the search for a solution by the selection and application of operators thought likely to remove differences between object structures.

Each node in the tree is essentially an independent definition of a particular subproblem. The root node defines the original problem supplied by the user. The nodes contain packed information such as the name of the current object, i.e. the object resulting from a particular sequence of operator applications, the name of the desired (goal) object, an ordered list of operators relevant to reducing differences between the objects, the value of the node, the best successor of the node, the level of the node, the operator which generated this node, etc., as well as linkage information. Nodes are linked by a pointer to the parent node, a pointer to the first son and a pointer to a brother node (Fig. 3.11):

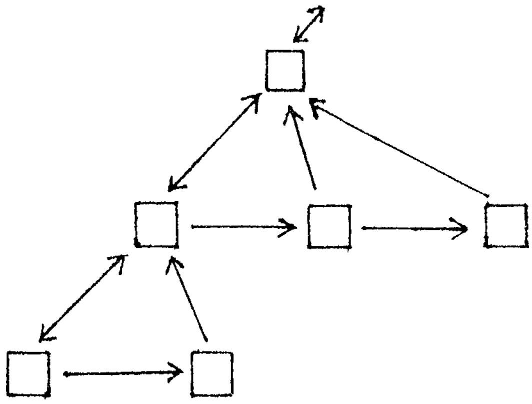


Figure 3.11

For each node the subproblem is to reduce the current object to the desired object.

If an operator can be directly applied to the current object at a node, a new node is generated as the son of the node under consideration. This node has the same goal object as the parent node but the current object is the result of applying the selected operator to the current object of the parent node.

If the selected operator cannot be applied directly a new son is generated containing the same current object as the parent but the new goal object is constructed in such a way that solution of the subproblem defined by the node will transform the current object to a state in which the operator is applicable. Assuming the object is to make operator $f_{ij}$ applicable, the goal is constructed recursively as follows.

Let $O_n$ mean any operator of degree n - in effect this is a variable with degree. Given any node j in the object structure, let h(j) be a function which returns a value m if j is the m'th son of the parent node. A goal object t is to be constructed. The following algorithm is performed:

(1) set t = input object of operator i; k = j.

(2) If k is the root node, exit.

(3) Set m = h(k), k = parent (k).

(4) Let $\ell$ be the highest index of a free variable in t and let the degree of k be n. A tree T is constructed s.t. the root node is $O_n$ and the ordered sons are $V_{\ell+1}, \ldots, V_{\ell+m-1}, t, V_{\ell+m+1}, \ldots, V_{\ell+n}$.

(5) Set t = T and go to (2).

At completion of the algorithm the structure t is of essentially the same shape as the current object. The structure rooted at node j of the current object corresponds to the input structure $I_i$ of operator i. All other non-terminal nodes in t are variable operators corresponding to the equivalent operators in structure s and all other terminal nodes are free variables corresponding to elements of s. The only differences detected will thus be between s(j) and $I_i$.

e.g. given rule $f_1$: $V_1 + V_2 := V_2 + V_1$.

Current object Fig. 3.12(a). If the aim is to apply $f_{12}$ to Fig. 3.12(a), the goal structure will be as in Fig. 3.12(b):


Figure 3.12

The depth of search in terms of the number of levels of subgoals generated in the attempt to make an operator applicable is limited by a user supplied parameter n. The top node is given the subgoal level of n. If a subgoal is generated it is given level n-1, and if a subgoal of this subgoal occurs it has level n-2 and so on. If a successor node is reached by the direct application of an operator it is given the same level as its parent. If a subgoal is developed with a level of less than zero it is ignored. A distinction must be drawn between the subgoal level of a node and the depth of a node. The subgoal level is the number of subproblems the system has 'looked ahead' in order to make an operator applicable. The depth of any node n is simply the number of nodes on the path from the top node to node n and is defined as the depth of its parent plus one. The top node has depth one.

When a new node is generated it is necessary, in order to prevent cycling, to determine whether the particular subproblem has been attempted previously. The testing is done by holding all previously generated object pairs. By filing each structure in the canonical tree it can be determined whether a structure has occurred before. If both the current structure and the goal structure of the node are not new, a binary search is employed to isolate the current object in the list of generated first members of the object pairs. The goal object is then compared with a linked list of goal objects allied with the particular initial object. Comparison is by canonical name.

If the pair has occurred previously at a depth much greater than that of the newly generated node the subtree rooted at this node is transferred to the new node as a shorter path to a goal is now possible. If the matched pair is at a depth less than or equal to the depth of the current node, the current node is simply deleted and the next best node selected for expansion.

When a new node has been generated it is necessary to detect the differences, if any, between the current object and the goal object. If there are any differences a (possibly empty) list of operators relevant to reducing the differences are generated and linked to the node. If there are no differences the current object is a specification of the goal object and the subproblem is solved. If the goal object is in fact the original goal the entire problem is solved - the node is marked and a backing up procedure applied to isolate the solution path.

If the goal object is not the top goal it is necessary to select the operator which generated the particular subgoal. This is done by backing up through the tree to the point at which the subgoal was first set up. This operator is then applied to the current object and a new node is generated to contain the result. As the subproblem has been solved the subgoal level of this node is incremented by one. The goal object is then that which was aimed for immediately before the subgoal was generated and is obtained from the parent node of the original subgoal.

The new node is then put through the same sequence of difference detection, selection of operators, etc.

## 3.10 LIMITATION OF OPERATORS

For efficiency in terms of time and space it is necessary to attempt to restrict the set of operators attached to each node as far as possible without eliminating those operators necessary for a solution. This restriction is achieved in two ways. Firstly by limiting the number of levels of difference and hence levels of operator by a given parameter (section 3.7) and secondly by keeping track of the purpose of subgoals.

When a subgoal is originally established it is in effect an independent subproblem. As a result it has no knowledge of the original differences which the operator would remove and little knowledge of the position the operator is to be applied to. It is necessary for the subproblem to be viewed in terms of some global strategy rather than in isolation as the danger arises that in transforming a structure to match the subgoal the final application of the operator may not remove the differences it was originally intended to.

When operators are selected by examining the second analysis table it is on the basis that some element of the structure s would remove a difference if transferred to the position corresponding to the difference. Such elements are considered 'essential elements' of the operator. If during transformation of the object the position or value of such elements is altered, application of the original operator would no longer remove the difference. The position at which the operator is to be applied must be held constant for the same reason.

Each subgoal thus contains two additional items of information, viz. the position of the operator which gave rise to the subgoal and the position of its 'essential element'. One or both of these may be empty: an operator may be to be applied to the root node in which case no transformation could alter its position and an operator may be selected from information in the first analysis table, i.e. the difference is removed by fixed constants in the table. The information is held as a packed linking structure showing the relationship to the root.

Any operator generated below a subgoal is tested to determine whether it destroys the purpose of this or any higher subgoal. Any such operators are deleted.

## 3.11 THE EVALUATION AND SELECTION OF NODES

In order to select any particular node for expansion it is necessary for each node to have some value indicative of its potential worth. Each node has an ordered set of operators, together with their values, attached to it. The node value is determined by a function of the n best operators at the node together with factors based on the depth of the node in the tree and the level of the node in terms of subgoals. n is a user-supplied parameter - if there are less than n operators then only these operators are considered.

As the operators are to some extent ordered so that operators which can be applied directly are favoured over those which require modification of the object, the tendency is to favour nodes which do not give rise to new subgoals.

The depth and subgoal level factors tend to favour those nodes nearer the root of the tree i.e. to add a breadth-first dimension to the search and those nodes which tend to be in the upper subgoal level. Nodes whose operator list is exhausted have value zero.

Every node in the tree contains the name of its best successor - if the node itself has a greater value than any successor it is considered its own best successor. When a node n is expanded it is re-evaluated in terms of the reduced operator list. Its new successor node m is also evaluated and the best successor of node n is selected. A backing-up procedure then alters, if necessary, all best successor names on the path from node n to the root node; if at any stage no alteration is necessary the procedure is halted. Only this path need be considered as all other nodes in the tree retain their best successor values.

A new node is initially given some user-supplied bias value to allow the system to force the search to some extent to follow a current path of solution before selecting another node. The bias decreases with increasing depth on a path.

The best successor of the top node is then the best node in the tree and is selected for expansion. If there is no best successor the problem is unsolvable.

The backing-up procedure is similar to that of MULTIPLE [27]: the major difference being that any node in the tree may be selected whereas MULTIPLE only deals with tip nodes.

## 3.12 OUTPUT OF RESULTS

The problem is solved when there are no differences between the current object and the original goal. In this case a backing-up procedure stacks the sequence of transformations from the goal node back to the root and outputs these in the correct order together with the series of operators applied.

The problem is unsolvable if there are no nodes containing operators left. The procedure is also halted if the maximum time specified by the user is exceeded.

# 4. A FORMAL APPROACH

## 4.1 INTRODUCTION

A formal approach is considered in an attempt to clarify the conditions under which the SDPS algorithm would be successful or unsuccessful. Ernst {8} has derived sufficient conditions for the success of the GPS algorithm. These conditions depend, however, on the ability to establish a fixed ordering on the static set of differences and on an explicit linking of the operators and differences. Ernst has noted that if a 'triangular' table of connections can be established convergence of the algorithm is assured.

Banerji {2} has developed a similar model and has derived a series of axioms under which a GPS-like algorithm will achieve success.

Both of these approaches are too inflexible to fit the SDPS model and the approach of this chapter will be merely to note the conditions under which the solution to a problem can be derived from the SDPS concept of differences. The model of a problem used is based on a general type called a W-problem by Banerji.

## 4.2 THE MODEL

A W-problem is a triple $<S, F, T>$ where S is a set of situations (states), T a subset of S called the goal states and F a set of partial functions on $S \times S$. The set of situations to which an operator $f_{ij}$ is directly applicable is denoted by $S_{f_{ij}}$.

Given a W-problem and an initial state $s^o \in S$ a solution sequence for $s^o$ is a sequence of functions $\{f_{i_1j_1}, f_{i_2j_2}, \ldots, f_{i_nj_n}\}$ such that $f_{i_k} \in F$ for each i and
$f_{i_nj_n}(f_{i_{n-1}j_{n-1}}(\ldots f_{i_1j_1}(s^o)\ldots))) = s^n \in T$.

The length of the solution is n. To simplify matters the notation $f_m$ for $f_{i_mj_m}$ will be used where no confusion could be caused.

The general aim in constructing a problem solver is to select some strategy for the construction of a solution sequence. Most heuristic strategies of this type are concerned simply with the selection of the next operator to be applied; see e.g. Nilsson {21}. However, a strategy for a subgoal building algorithm must have the ability to 'look ahead' for operators to be applied later in the sequence, and to select operators relevant to reaching a state in which these operators may be applied.

The first concept to be defined is that of distance from a goal. A state s is at a distance i from a goal state if there exists a solution sequence for s of length i.

Let $s^iBs^j$ mean $s^j = f(s^i)$ for some $f \in F$. Let B' be the transitive closure of B, i.e. if $s^i B' s^j$ then the state $s^j$ can be reached from $s^i$ by a finite sequence of operator applications.

Let $G_{jk}$ be any particular sequence s.t. $s^j B' s^k$ and let $G'_{jk}$ be the set of all such sequences.

A set $T_i$ is defined as the set of all states s of distance i from the goal state t such that the sequence of operators will not reproduce s in some $T_k$, $k < i$.

Formally, let $T_o = t$ and for $i > o$.

$$T_{i+1} = \{s \mid (\exists f)(f \in F \ \& \ f(s) \in T_i) \text{ and } \nexists \ G_{o,i+1} \in G'_{o,i+1} \text{ such that a subsequence } G_{o,i-k+1} \text{ will reproduce } s \text{ in } T_k \text{ for any } k < i+1\}.$$

Obviously the sets are not disjoint but cycling is avoided by the second condition.

## 4.3 DIFFERENCE-DERIVABLE SOLUTIONS

Although it is possible to consider problem solving strategies based on the sets $T_i$ more flexibility is required to consider both the concept of subgoals and the idea of a strategy based on differences.

Rather the set of states is considered for which a difference-derivable (DD) solution of some length exists. Formally the set $V_i$ consists of those states s of length i from the goal such that $s \in T_i$ and such that a DD solution exists for each s. Obviously $V_i \subseteq T_i$. As any problem may have a number of DD solutions the sets are not disjoint.

To handle the concept of subgoals it is necessary to consider the solution of problems in which the goal is not the original t. In SDPS a subgoal is used to transform a state to one in which a specific operator is applicable. If $f_{ij}$ is to be applied then the current state $s^m$ must be transformed to the set of states in which $f_{ij}$ is applicable. This set of states is denoted by $S_{f_{ij}}(s^m)$.

A new set of states $Z_i$ is introduced. These are those states of distance i from a goal $Z_o = S_{f_{ij}}(s)$ for which a solution sequence is DD. If $Z_o = t$ then $Z_i = V_i$ for all i.

Note that given some $s^m \in Z_i$ and a DD solution sequence $f_i(f_{i-1}(\ldots(f_1(s^m))))$ then $f_1(s^m)$ does not necessarily belong to $Z_{i-1}$. This is due to the possible occurrence of additional subgoals in deriving the sequence.

Let the ordered pair $< s^o, t >$ represent the problem of transforming $s^o$ to t. A solution to the problem could be considered as an ordered sequence of operator applications represented by an ordered n-tuple $G = (f_1, f_2, \ldots, f_n)$ (where $f_k = f_{i_k j_k}$) such that $f_n(\ldots f_1(s^o)) = s^n$ S t. Note that $f_1(s^o) = s^1$, $f_2(s^1) = s^2$, etc.

Let $F^i(X)$ denote the set of operators discovered by the SDPS method given state $s^i$ and goal X. A solution G to a problem is difference-derivable (DD) if either

(a) $s^o$ S t, i.e. $s \in T_o$, or

(b) $\exists$ some $f_k \in G \cap F^o(t)$ such that the ordered solution $(f_1, f_2, \ldots, f_{k-1})$ to the problem $< s^o, s_{f_k}(s^o) >$ is DD and for some particular $s^{k-1} \in S_{f_k}(s^o)$ the solution $(f_{k+1}, \ldots, f_n)$ to $< f_k(s^{k-1}), t >$ is DD. ($s^{k-1}$ is the result of the correct solution to the first problem.)

i.e. if $Z_o = S_{f_k}(s^o)$ then $s^o \in Z_{k-1}$ and if $Z_o = t$ then $f_k(s^{k-1}) = s^k \in Z_{n-k}$.

The ordering of the solution to the subproblems is essential - if it is solved by another sequence the resulting $s^{k-1}$ may not be the correct state in the context of the entire problem.

Loosely the definition implies that for each node $< s^m, X >$ on a solution path in the goal tree S DPSmust have in the set of operators either $f_{m+1}$ or some $f_p$ in the correct sequence. At each state in the path SDPS must at some stage be able to select the correct operator to be applied to that state, i.e. given $s^m$ the differences between $s^m$ and the current goal must at some depth of subgoal generate $f_{m+1}$. This operator is obviously only valid if the current goal is on a correct path.

For each state $s^m$, (m = o,...,n-1) which is correctly attained on a DD path we may consider the state $s^{m+1}$ as derivable from $< s^m, Z >$ for some goal Z if one of the following holds:

(a) $s^m \in Z_o = S_{f_{m+1}}$

(b) $f_{m+1} \in F^m(Z)$

(c) $\exists$ some $f_k \in F^m(Z) \cap G$ such that $s^m \not\in s_{f_k}$ and the state $s^{m+1}$ is derivable from $< s^m, s_{f_k}(s^m) >$.

A solution to a problem $< s, t >$ is thus obviously not DD if at any stage it is not possible to generate the correct successor to a state.

In terms of the concept of DD solutions it may be informative to reconsider the conditions under which a particular operator $f_{ij}$ is selected.

Zero-level differences are selected by the method of section 3.5. The higher level differences are generated as described in 3.7.

For any operator $f_i$ let I denote the input structure and O the output structure. For any two structures s and t, let $s_k$ L $t_m$ mean that element $s_k \in s$ loosely corresponds or matches to element $t_m \in t$. This concept of loose correspondence need not be the same as that of SDPS.

To select a particular operator $f_{ij}$ it is necessary that some difference (t', k) exist such that one of the following three conditions holds. Any reference to $I_\ell \in I$ will imply that $I_\ell \subset s_j$, i.e. the matching is against substructure s(j).

1. (a) $I_\ell$ L $S_k$ for some $\ell$;
   (b) $I_\ell$ L $O_m$ for some m;
   (c) $O_m \in C$;
   (d) $O_m$ s t'.
   i.e. there exists some constant symbol in O which is equivalent to t'.

2. (a) $I_\ell$ L $S_k$ for some $\ell$;
   (b) $I_\ell$ L $O_m$ for some m;
   (c) $O_m = V_a$;
   (d) $\exists$ q s.t. s(q) $\subseteq$ s(j) and $s_q$ L $I_p$ for some p s.t. $O_m = I_p = V_a$;
   (e) $s_q$ S t'.

3. (a) $I_\ell$ L $S_n$ for some $\ell$, n and s(k) $\subseteq$ s(n);
   (b) $I_\ell$ L $O_m$ for some m;
   (c) $\exists$ r s.t. s(r) $\subseteq$ s(j) and $S_r$ L $I_p$ for some p and $O_m = I_p = V_a$;
   (d) $\exists$ q s.t. s(q) $\subseteq$ s(r) and $S_q$ L $S_k$ for structures rooted at r and n respectively;
   (e) $s_q$ S t'.

The particular difference (t', k) may be either a zero-order difference or it may be generated from such a difference (t', m) by the procedure outlined below.

If the $s_q$ generated by 2 or 3 above is not a specification of t', a new difference (t', q) is generated. If q = k the correct difference has been generated. If $q \neq k$ it is necessary that there exist some operator which will generate a new difference (t', r) using difference (t', q), and so on. To generate (t', k) it is thus necessary that there exist a sequence of operators $(f_{i_1 j_1},\ldots,f_{i_b j_b})$ such that given a zero level difference (t', m) a set of progressively higher order differences $(t', r_1)$, $(t', r_2)$,... $(t', r_b)$ is generated and $r_b = k$. Each new difference $(t', r_{k-1})$ serves as input to the operator $f_{i_{k-1} j_{k-1}}$ to generate $(t', r_k)$.

Both the outline of the recursive definition of DD solutions and the generation of higher order differences take no note of the limitations placed on these in SDPS by limiting the depth of subgoals and the number of differences allowed. These restrictions are purely for efficiency and do not detract from the basic definition.

To illustrate the concept of DD and non-DD solutions three simple examples are considered. The examples are selected from the area of propositional calculus.

Example 1

A solution path for which no subgoals are necessary, i.e. the algorithm will determine the correct operators to be applied to each state immediately.

The operators are:

D1 : $v_1 \Rightarrow v_2 := \sim v_1 \vee v_2$.

D2 : $v_1 \vee v_2 := v_2 \vee v_1$.

The operator representation is in Figs. 4.1(a) and (b).

The problem is to prove that

$$A \Rightarrow (B \vee C) \; := \sim A \vee (C \vee B)$$

- the representation of the input and goal structures are figures 4.2(a) and (c) respectively.

The solution is $(f_{11}, f_{24})$. The initial difference detected is (v,1) and operator $f_{11}$ is the only operator capable of transforming ⇒ to v so is immediately applied, giving the result in 4.2(b). The differences selected between 4.2(b) and 4.2(c) are (B, 6) and (C, 5), which again $f_{24}$ will remove immediately.

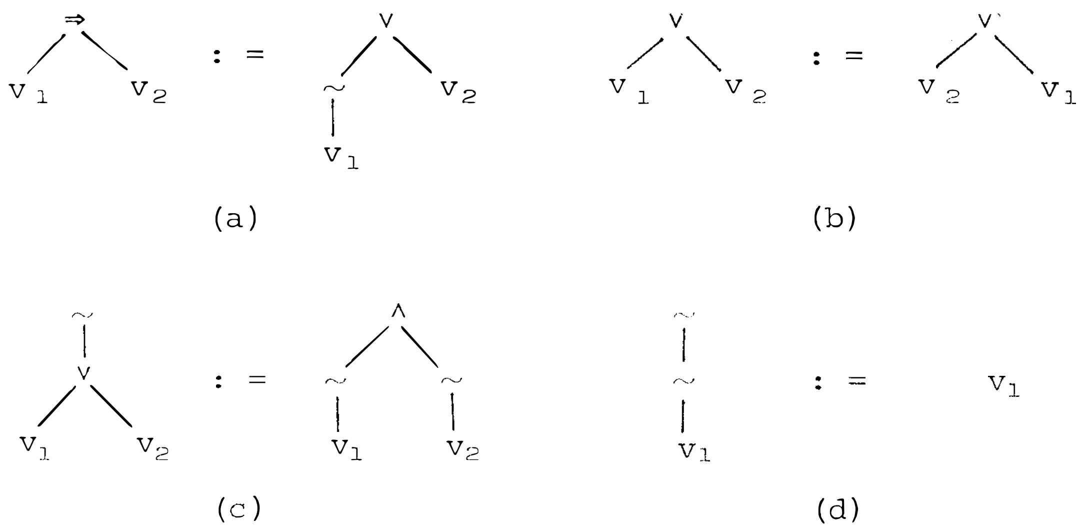



Figure 4.1

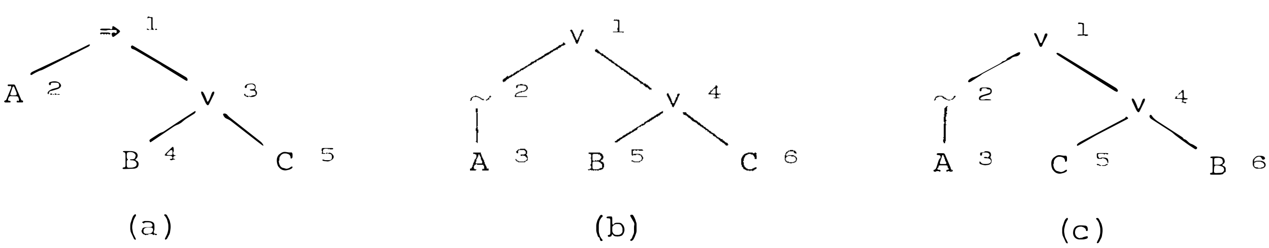



Figure 4.2

Example 2

A problem showing the use of subgoals to derive a solution.

The operators are:

D1 : $v_1 \Rightarrow v_2$ : = $\sim v_1 \vee v_2$. Fig. 4.1(a)

D3 : $\sim (v_1 \vee v_2)$ : = $\sim v_1 \wedge \sim v_2$. 4.1(c)

D4 : $\sim \sim v_1$ : = $v_1$ 4.1(d)

The problem is to prove $\sim (A \Rightarrow B)$ : = $A \wedge \sim B$ ; the initial and goal structures are given in Fig. 4.3(a) and (d) respectively. The solution sequence is ($f_{12}$, $f_{21}$, $f_{32}$).

The initial difference detected is ($\wedge$, 1). $f_{21}$ is the only operator capable of transforming $\sim$ to $\wedge$ but it cannot be directly applied. A subgoal of attaining a state equivalent to the input of $f_2$ is established, i.e. the goal is the input structure in Fig. 4.2(b). The difference between this goal and 4.3(a) is ($\vee$, 2). Rule $f_{21}$ will remove this difference and is applied, giving 4.3(b) which is now a specification of the subgoal. Application of $f_{21}$ then gives 4.3(c) and the difference between 4.3(c) and 4.3(d) selects $f_{32}$, giving the desired result.

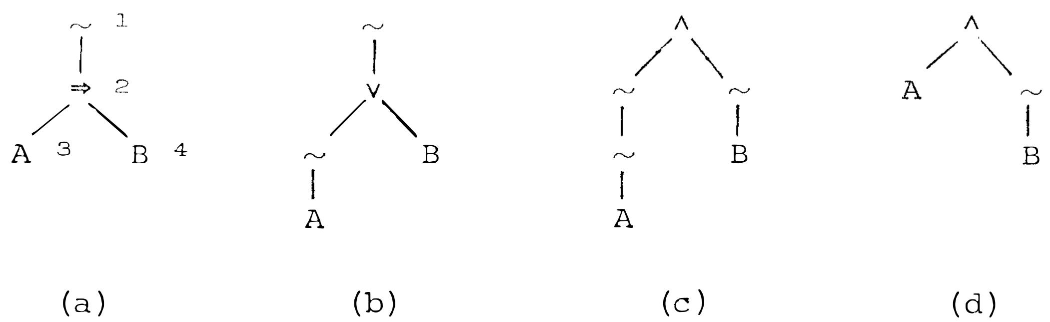



Figure 4.3

Example 3.

A non-DD solution.

The operators are:

D2 : $v_1 \vee v_2 \quad := v_2 \vee v_1$.

D3 : $\sim (v_1 \vee v_2) := \sim v_1 \wedge \sim v_2$.

and the problem is to prove $\sim (B \vee A) := \sim A \wedge \sim B$; the initial and goal structures are Figs. 4.4(a) and (c) respectively.

The solution is $(f_{22}, f_{31})$. The initial difference selected is $(\wedge, 1)$ and the only operator capable of removing it is $f_{31}$. However $s^1 \in S_{f_{31}}(s^1)$ and $f_{31}$ may be applied immediately, giving Fig. 4.4(b). The differences here are (A, 3) and (B, 5) but there is no operator capable of removing them. There is no way that SDPS can detect from the formal definition of differences that $f_{22}$ must be applied before $f_{31}$.

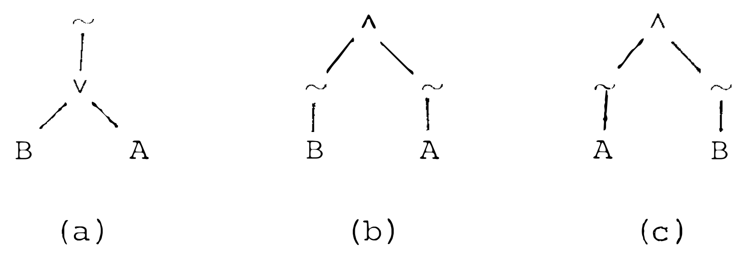



Figure 4.4

## 4.4 THE SDPS ALGORITHM

The problem solving steps taken by SDPS are summarized in the algorithm set out below. Let son (k) denote a note which is a direct successor of node k; parent (k) denote the parent of k. Let $(s^k, t^k)$ refer to the current state $s^k$ and the goal $t^k$ at node k in the tree. Let level (k) refer to the subgoal level.

(1) Set $s^1 = s$, $t^1 = t$, $k = 1$, level (k) = max. subgoals. Generate first set of operators.

(2) If there are no expandable nodes left or if maxtime is exceeded, exit with failure.

(3) Select best operator $f_{ij}$ at node (k).

(4) If $s^k(j)$ S $I_i$ (i.e. operator can be applied immediately) then generate new node n = son (k) with $(f_{ij}(s^k), t^k) \in n$, level (n) = level (k) and go to (6).

(5) If level (k) is such that a new subgoal (n) would have level (n) $<$ o then go to (11). Otherwise set up node (n) = son (k) with $(s^k, S_{f_{ij}}(s^k)) \in n$. level (n) = level (k)-1.

(6) If $(s^n, t^n)$ is not new, delete node n and go to (11).

(7) Generate differences. If there are differences generate a set of operators, file these and go to (10).

(8) If $t^1 = t^n$ exit with solution. Else find $f_{ij}$ at node m which generated this subgoal.

(9) Set up node $\ell$ = son (n) with $(f_{ij}(s^n), g^m) \in \ell$, level($\ell$) = level (m), n = $\ell$. Go to (6).

(10) Evaluate node n.

(11) Select best node i in the tree. k = i. Go to (2).

The algorithm will find a DD solution of finite length if one exists, subject to the constraints of maximum time and the practical considerations of the maximum depth of subgoals and level of differences allowed.

Cycling is prevented by step (6). If a correct solution is obtained an exit is made from step (8) and failure is admitted at step (2). To ensure that all nodes within a certain depth in the tree will be searched the depth of the node is used as a factor in evaluating the node, and carries decreasing weight with greater depth. This prevents too deep a search beyond the limits of a probable solution as the factor will eventually lower the value of any deep node sufficiently to allow any shallower nodes to be expanded.

# 5. RESULTS AND CONCLUSIONS

## 5.1 EVALUATION OF PERFORMANCE

To determine the efficiency of a given heuristic technique it is necessary to establish some measures of performance of the system. The criterion of time-to-solution is rather too dependent on extraneous factors such as language of implementation, machine used, etc., and measures are required which show how well the search is directed towards a goal in terms of the shape of the problem solving tree. Two such measures are penetrance (P) and effective branching factor (B) [6, 21].

If L is the number of nodes on a solution path attained by direct application of an operator plus the initial node and T is the total of such nodes in the tree then the penetrance P is defined by

$$P = L/T$$

The definition ignores those nodes which simply define new subgoals. This is in order to allow comparison with those systems which do not use a subgoal concept.

Penetrance as a measure of efficiency varies with the difficulty of the problem as well as the efficiency of the search method and is thus only really useful for comparing problems of a similar standard.

The definition of the effective branching factor, B, is based on the concept of a tree equal in depth to the solution path length and having a total number of nodes equal to the number generated during search. B is then the constant number of successors that would be possessed by each node in the tree. In SDPS all nodes in the tree are considered. If M is the number of nodes in the solution path and Q the total number in the tree then the effective branching factor is defined by

$$\frac{B}{(B-1)} (B^M - 1) = Q$$

B cannot be calculated directly for given values of M and Q. To overcome this problem in SDPS a large number of values of Q were calculated for successive increments of M and a range of values of B for each M. Using Lagrangian interpolation it is possible to derive values of B for integral values of Q, given some value for M. A large table of such values is held in a disk file to be indexed for the particular values of Q and M resulting from the solution of a problem.

## 5.2 SOME EXAMPLES

Appendix A contains eight examples of the type of problem solved by the SDPS system. Each example is discussed briefly below and the notation used is outlined. The examples specify the particular operators presented to SDPS in the form of the first line giving the input structure and the second the output structure. The solution sequence of transformations is given with the operators applied to attain each new state and the time taken to achieve solution, the penetrance and the effective branching factor (EBF) are also included.

The routine which translates from the internal representation to some standard external form assumes a left-to-right sequence of evaluation so that operators of equal precedence do not have the left-most set of brackets inserted. For this reason an expression which may in the context of the problem be most naturally represented by e.g. $(x + y) + z$ will appear in the listing as $x + y + z$.

Most of the examples given have been solved by other problem solving systems. However any comparison for problems solved by FDS can only be on the basis of a time-to-solution criterion of efficiency. The figures achieved for SDPS may in certain cases suffer from the fact that on the Univac 1106 system at U.C.T. the time taken to solve any particular problem may vary with the user load on the machine. As GPS uses the four types of goal as opposed to the one of SDPS no simple comparison on the basis of any empirical measurement can be made. However it is worth noting that for problems solved by both SDPS and GPS the formal concept of differences is sufficient to determine solutions in certain cases which in GPS requires the explicit operator/difference linking supplied by the table-of-connections.

### 5.2.1 PARSING SENTENCES

Generative grammars of certain languages may be defined by a set of phrase-structure rules. Words of the language are divided into classes called parts of speech. The rules of the language may be used as operators to parse sentences to determine whether they belong to the language or not.

The rules of the particular language presented to SDPS for this problem are:

(1) NP VP NP : = S

(2) NP VBP AP : = S

(3) AP < adjective > : = AP

(4) < adjective > : = AP

(5) AP < noun > : = NP

(6) < noun > : = NP

(7) < adverb > < verb > : = VP

(8) < verb > : = VP

(9) < adverb > < verb-be > : = VBP

(10) < verb-be > : = VBP

The symbols used are defined as:

S Sentence

NP Noun phrase

AP Adjective phrase

VP Verb phrase

VBP Verb phrase for-to-be.

To specify the operators for SDPS a linear connective of second degree (.C.) is introduced to order the constituent phrases of each rule, e.g. rule (1) above is represented as

NP.C. VP.C. NP : = S.

The problem given was to parse the sentence:

Free variables cause confusion.

A set of terminal classes is defined for adjectives, nouns, etc., and each word in the sentence is defined as belonging to its specific class, i.e. 'Free' belongs to the class of adjectives, 'variables' to the class of nouns, etc.

Both the penetrance and the EBF show a fairly direct and efficient solution of the problem. The problem is a good example of the close relation between the SDPS operators and compiler productions.

The problem is identical to one of these presented to GPS {9}. GPS also found a fairly direct proof involving 19 goals but did of course require the explicit linking of operators and differences.

### 5.2.2 EIGHT-PUZZLES

The 8-puzzle is one of a large class of sliding block puzzles and has been widely used as an example in problem solvers, particularly those employing a state-space approach {6, 20}. It consists of eight numbered, movable tiles set in a 3 × 3 frame. One cell of the frame is always empty, making it possible to move an adjacent tile into the empty cell.

The configuration may be conveniently represented by a 3 × 3 matrix using a zero to designate the empty space. Twenty-four operators are necessary for the SDPS formulation, each having the form, e.g.

$$\begin{matrix} V_1 & V_2 & V_3 \\ V_8 & V_4 & O \\ V_7 & V_6 & V_5 \end{matrix} \quad := \quad \begin{matrix} V_1 & V_2 & V_3 \\ V_8 & O & V_4 \\ V_7 & V_6 & V_5 \end{matrix}$$

Two problems were given to SDPS, one requiring five transformations to achieve the goal and one requiring ten. On the shorter puzzle SDPS proved very efficient, achieving a penetrance of 0.714 and an EBF of 1.091. For the identical problem Nilsson {21} obtained results of P = 0.108, B = 1.86 for breadth-first search and P = 0.385, B = 1.34 for a state-space search using a simple evaluation function.

On the longer problem SDPS did not do so well and a trace showed that this was due to a tendency to lose its way near the base of the problem solving tree. Search tended to be rather random until a reasonable distance from the goal was achieved. Lengthening the look ahead factor had no real effect on this tendency.

### 5.2.3 BOOLEAN ALGEBRA

The problem is taken from Modern Applied Algebra (G. Birkhoff and T.C. Bartee) {3}.

A Boolean algebra may be defined as a set A with two binary operations ($\wedge$, $\vee$), two universal bounds (O, I) and one unary operation ' such that a given set of axioms hold for all $x, y, z \in A$.

The following subset of the axioms were given to SDPS as operators:

| | | | |
|---|---|---|---|
| (1) | $x \wedge x = x$ | $x \vee x = x$ | (Idempotent) |
| (2) | $x \wedge y = y \wedge x$ | $x \vee y = y \vee x$ | (Commutative) |
| (3) | $x \wedge (y \wedge z) = (x \wedge y) \wedge z$ | $x \vee (y \vee z) = (x \vee y) \vee z$ | (Associative) |
| (4) | $x \wedge (x \vee y) = x$ | $x \vee (x \wedge y) = x$ | (Absorption) |
| (5) | $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$ | $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$ | (Distribution) |

The symbols $\wedge$, V are called 'wedge' and 'vee' respectively. The problem given to SDPS is to prove Lemma 2 in the reference (page 131), i.e. that the axiom of Modularity may be derived from the given axioms where the axiom of Modularity is defined as:

$$x \wedge \{y \vee (x \wedge z)\} = (x \wedge y) \vee (x \wedge z).$$

SDPS achieved the solution in 38 seconds with a penetrance of 0.139 and an EBF of 1.267. Although the solution path is fairly short the system appeared to have some difficulty in selecting operators. However the problem does show that SDPS is capable of solving problems which humans find fairly difficult.

### 5.2.4 PROPOSITIONAL CALCULUS

The operators for these problems are a set of legitimate transformations of propositional calculus of the form e.g.

$$\sim A \wedge (B \Rightarrow A) \; : = \; \sim B$$

Five problems were given to the system in the same form, e.g. Prove that $(A \Rightarrow \sim B) \wedge B$ is equivalent to $\sim A$.

As each problem was proved it was added to the set of operators as a theorem. As logical notation is not available on the printout, the words AND, OR, NOT and IM were used for $\wedge$, V, $\sim$, $\Rightarrow$ respectively.

The solutions are fairly direct and the SDPS system works very efficiently here. The same set of problems and operators were presented to FDS [22] and in terms of a time-to-solution criterion SDPS and FDS have roughly the same efficiency.

The algorithm performs well as each proof has a direct transformation property in that each line of the proof is achieved by the direct application of an operator to the preceding line. More general proofs in propositional logic which require flexible use of the rule of detachment (i.e. given A and A ⇒ B, infer B) cannot be easily specified in SDPS as these proofs essentially involve the manipulation of a set of inferred clauses. This implies that sub-proofs would have to be obtained independently and linked together at later stages of the proof sequence. As the operation of SDPS is inherently sequential and each node completely defines one complete state achieved with its current goal, this linking of subproblems does not appear to have a simple solution.

For the same reason the proof of predicate calculus theorems using the resolution principle as an operator is not feasible in the SDPS format.

### 5.2.5 ELEMENTARY ALGEBRA

Six standard rules for the manipulation of simple algebraic expressions are specified as operators in the form e.g. $X + Y := Y + X$ .

The theorems to be proved are given as simple algebraic statements of the form:

prove that $(x - y) + y$ is equivalent to $x$.

Solution then involves manipulating the input expression with the given operators until the goal expression is achieved. As each theorem was proved it was added to the set of operators.

The proofs are in general fairly direct and SDPS performs of the river and whether the father is present or not, and a unary operator BOAT which determines whether the boat is on the left or right bank. As SDPS has no concept of simple arithmetic, the addition and subtraction of the sons must be explicitly stated by the operators.

SDPS achieves a solution in seven seconds and the fairly low penetrance and high EBF show that search is not exceptionally well directed. GPS solved the identical problem in 33 goals.

5.2.7 A LOGIC PUZZLE

The following formulation of a logic puzzle is taken from one presented to FDS {22}.

> There are two opponents, Ed and Al, each of whom either always tells the truth or always lies. A philosopher approaches the pair and asks if the library is to the east or west. Ed mutters something and Al states "Ed says east but he's a liar". In which direction is the library?

SDPS uses the following sets of constant symbols:

$C_2$ = {SAYS, IS, IM (implies), EQ (equivalent), AND}.

$C_1$ = {NOT}.

$C_0$ = {TTLR (truthteller), LIAR, EAST, WEST, DIRN direction), AL, ED, DATA}.

EAST, WEST and DIRN are declared as belonging to the same constant class.

The SDPS operators and the problem specification are given in Appendix A together with the solution found. SDPS obtains a solution in 32 seconds which is rather slower than the FDS solution but the penetrance and EBF indicate a reasonably well-directed search.

### 5.2.8 THE MONKEY AND BANANAS PROBLEM

The monkey-and-bananas problem is often used in artificial intelligence to demonstrate the operation of problem solvers designed to perform reasoning {9}. The problem can be stated simply as follows:

> A room contains a monkey, a box and some bananas hanging from the ceiling out of reach of the monkey. The bananas can only be reached when the monkey is standing on the box when it is under the bananas. What sequence of actions will allow the monkey to get the bananas?

The SDPS formulation uses the following sets of constants:

$C_2$ = {AND, AT (position of)}.

$C_1$ = {NOT}.

$C_0$ = {MON (monkey), ONBOX (monkey is on the box), BOX, HB (monkey has bananas), A, B, C (positions in the room)}.

The solution achieved by SDPS is direct which it should be with the limited possibilities offered by the operators.

### 5.2.9 OTHER PROBLEMS

SDPS has been applied to a number of similar problems. In most cases solutions were achieved with results similar to those above and in certain cases no solution could be obtained in the time allowed. No problems of this type were found which failed due to an inability to discover a difference-derivable path provided a sufficiently general problem representation was selected.

## 5.3 CONCLUSIONS

The SDPS system performs well on a certain set of problems whose proof sequences are characterised by the direct transformation property in that each new state may be derived directly from the previous state by the application of a single operator. SDPS has achieved the solution of problems which humans may find relatively difficult. On problems with a short solution path the use of differences is sufficient to find efficient proofs but on longer problems there is an obvious drop in efficiency. This tendency is found in most problem solvers employing tree search as in general there is some difficulty in establishing the first stages of a long solution path. As a single general technique the formal difference heuristic functions very well but is obviously not as efficient as those systems using problem-specific heuristics.

As a large variety of problems can be formulated within the framework of the direct transformation property the system may be said to be general purpose. Questions on the effective generality of the difference technique were considered in Chapter Four. From the conditions noted there it can be seen that there are obviously problems for which SDPS would not be capable of attaining a solution. Although this is an obvious limitation on the system it would appear in practice that such problems are rare, given adequate formulation of the problem environment. In most cases considered SDPS obtained a solution although it need not find the shortest path or the most obvious solution.

Although the formal difference heuristic does not appear sufficiently powerful to solve 'complex' problems within a limited time it compares fairly well with the performance of other systems. It naturally suffers from the generality/ efficiency conflict discussed in section 1.3 and its most feasible use is probably in conjunction with the employment of problem-specific heuristics to enable a more accurately directed and hence deeper search within a more limited problem environment.

# BIBLIOGRAPHY

{1} AMAREL, S., "On representations of problems of reasoning about actions", in D. Michie (ed.) Machine Intelligence 3 pp. 131-171, American Elsevier Publishing Company, New York, 1968.

{2} BANERJI, R.B., Theory of Problem Solving: An Approach to Artificial Intelligence, in R. Bellman (ed.) Modern Analytic and Computational Methods in Science and Mathematics, American Elsevier Publishing Co., New York, 1969.

{3} BIRKHOFF, G. and T.C. BARTEE, Modern Applied Algebra, McGraw-Hill Book Company, 1970.

{4} BROWN, P.J., "Recreation of Source Code from Reverse Polish Form", Software - Practice and Experience, Vol. 2, No. 3, pp. 275-278, 1972.

{5} CHARLTON, C.C. and P.G. HIBBARD, "A Note on Recreating Source Code from the Reverse Polish Form", Software - Practice and Experience, Vol. 3, No. 2, pp. 151-153, 1973.

{6} DORAN, J.E. and D. MICHIE, "Experiments with the Graph Traverser Program", Proc. Roy. Soc. A, Vol. 294, pp. 235-239, 1966.

{7} ERNST, G.W., "GPS and Decision Making: An Overview", in R. Banerji and M.D. Mesarovic (eds.) Theoretical Approaches to Non-Numerical Problem Solving, Springer-Verlag, 1970.

{8} ERNST, G.W., "Sufficient Conditions for the Success of GPS", J.ACM., Vol. 16, No. 4, October 1969.

{9} ERNST, G.W. and A. NEWELL, GPS: A Case Study in Generality and Problem Solving, ACM Monographs, Academic Press, New York, 1969.

{10} FEIGENBAUM, E.A., "Artificial Intelligence: themes in the second decade", Proc. IFIP68, International Congress, Edinburgh, 1968.

{11} FEIGENBAUM, E.A., B.G. BUCHANAN and J. LEDERBURG, "On Generality and Problem Solving: A case study using the DENDRAL program", in B. Meltzer and D. Michie (eds.) Machine Intelligence 6, Edinburgh University Press, 1971.

{12} FIKES, R.E. and N.J. NILSSON, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving", Artificial Intelligence, Vol. 2, No. 3/4, pp. 189-208, Winter 1971.

{13} JACKSON, P.C., Introduction to Artificial Intelligence, Petrocelli Books, New York, 1974.

{14} KNUTH, D.E., Fundamental Algorithms - The Art of Computer Programming, Vol. 1, Addison Wesley, 1969.

{15} MESAROVIC, M.D., "A Systems Theoretic Approach to a Formal Theory of Problem Solving" in R. Banerji and M.D. Mesarovic (eds.), Theoretical approaches to Non-Numerical Problem Solving, Springer-Verlag, 1970.

{16} MICHIE, D., "Strategy Building with the Graph Traverser" in N. Collins and D. Michie (eds.) Machine Intelligence 1, Oliver and Boyd, Edinburgh, 1967.

{17} NEWELL, A. and H.A. SIMON, "GPS: A Program that simulates human thought" in E.A. Feigenbaum and J. Feldman (eds.) Computers and Thought , McGraw-Hill, 1963.

{18} NEWELL, A., J.C. SHAW and H.A. SIMON, "Empirical explorations with the Logic Theory Machine: A case study in Heuristics" in E.A. Feigenbaum and J. Feldman (eds.) Computers and Thought , McGraw-Hill, 1963.

{19} NEWELL, A., "Remarks on the Relationship between Artificial Intelligence and Cognitive Psychology", in R. Banerji and M.D. Mesarovic (eds.) Theoretical Approaches to Non-Numerical Problem Solving, Springer-Verlag, 1970.

{20} NEWELL, A. and H.A. SIMON, Human Problem Solving, Prentice-Hall Inc., 1972.

{21} NILSSON, N.J., Problem Solving Methods in Artificial Intelligence, McGraw-Hill, 1971.

{22} QUINLAN, J.R. and E.B. HUNT, "A Formal Deductive Problem Solving System", J.ACM., Vol. 15, No. 4, pp. 625-646, October 1968.

{23} QUINLAN, J.R., "A Task Independent Experience Gathering scheme for a Problem Solver", Proc. Intern. Joint Conf. Artificial Intelligence, pp. 193-198, 1969.

{24} SANDEWALL, E.J., "A planning Problem-Solver based on Look ahead in Stochastic Game Trees", J.ACM., Vol. 16, No. 3, July 1969.

{25} SANDEWALL, E.J., "Heuristic Search: Concepts and Methods" in N.V. Findler and B. Meltzer (eds.) Artificial Intelligence and Heuristic Programming, Edinburgh University Press, 1971.

{26} SIMON, H.A., "On Reasoning About Actions" in H.A. Simon and L. Siklossy (eds.) Representation and Meaning - Experiments with Information Processing Systems, Prentice-Hall, 1972.

{27} SLAGLE, J.R. and P. BURSKY, "Experiments with a Multipurpose, Theorem-Proving Heuristic Program", J.ACM., Vol. 15, No. 1, January 1968.

{28} SLAGLE, J.R., Artificial Intelligence: The Heuristic Programming Approach, McGraw-Hill, 1971.

---ooOoo---

# APPENDIX A

## SOME EXAMPLES OF SDPS OPERATION

```
@HDG PARSING PROBLEM ONE
@XQT
BEGIN EXECUTION. NU AACSOLLIBRARY LEVEL 6.2, FEB. 15, 1974

OPERATOR                    1

NP.C.VP.C.NP
S

OPERATOR                    2

NP.C.VBP.C.AP
S

OPERATOR                    3

AP.C.ADJ
AP

OPERATOR                    4

ADJ
AP

OPERATOR                    5

AP.C.NOUN
NP

OPERATOR                    6

NOUN
NP

OPERATOR                    7

ADV.C.VERB
VP

OPERATOR                    8

VERB
VP

OPERATOR                    9

ADV.C.VRBE
VBP

OPERATOR                   10

VRBE
VBP
```

```
------------------------------
 PROVE THAT

FREE.C.VARIABLES.C.CAUSE.C.CONFUSION

IS EQUIVALENT TO

S

------------------------------

SOLUTION TIME(SECS):                3

PENETRANCE     5.0000+-01

EFFECTIVE BRANCHING FACTOR:               1.0225+00

                                                              APPLY OP  0
FREE.C.VARIABLES.C.CAUSE.C.CONFUSION

                                                              APPLY OP  4
AP.C.VARIABLES.C.CAUSE.C.CONFUSION

                                                              APPLY OP  5
NP.C.CAUSE.C.CONFUSION

                                                              APPLY OP  6
NP.C.CAUSE.C.NP

                                                              APPLY OP  8
NP.C.VP.C.NP

                                                              APPLY OP  1
S

EXECUTION TIME      5.570 SECONDS
```

```
@HDG EIGHT PUZZLES PROBLEM 1
@XQT
BEGIN EXECUTION. NU ALGOL LIBRARY LEVEL 6.2, FEB. 15, 1974

OPERATOR                1

   1   2   3               1   2   3
   4   0   5     :=        0   4   5
   6   7   8               6   7   8

OPERATOR                2

   1   2   3               1   2   3
   0   4   5     :=        4   0   5
   6   7   8               6   7   8

OPERATOR                3

   1   2   3               1   2   3
   4   0   5     :=        4   7   5
   6   7   8               6   0   8

OPERATOR                4

   1   2   3               1   2   3
   4   7   5     :=        4   0   5
   6   0   8               6   7   8

OPERATOR                5

   1   2   3               1   2   3
   4   0   5     :=        4   5   0
   6   7   8               6   7   8

OPERATOR                6

   1   2   3               1   2   3
   4   5   0     :=        4   0   5
   6   7   8               6   7   8

OPERATOR                7

   1   2   3               1   0   3
   4   0   5     :=        4   2   5
   6   7   8               6   7   8

OPERATOR                8
```

```
    1   0   3            1   2   3
    4   2   5     :=     4   0   5
    6   7   8            6   7   8

OPERATOR           9

    1   2   3            0   2   3
    0   4   5     :=     1   4   5
    6   7   8            6   7   8

OPERATOR          10

    0   2   3            1   2   3
    1   4   5     :=     0   4   5
    6   7   8            6   7   8

OPERATOR          11

    1   2   3            1   2   3
    0   4   5     :=     6   4   5
    6   7   8            0   7   8

OPERATOR          12

    1   2   3            1   2   3
    6   4   5     :=     0   4   5
    0   7   8            6   7   8

OPERATOR          13

    1   0   3            0   1   3
    4   2   5     :=     4   2   5
    6   7   8            6   7   8

OPERATOR          14

    0   1   3            1   0   3
    4   2   5     :=     4   2   5
    6   7   8            6   7   8

OPERATOR          15

    1   0   3            1   3   0
    4   2   5     :=     4   2   5
    6   7   8            6   7   8

OPERATOR          16
```

```
    1   3   0              1   0   3
    4   2   5    :=        4   2   5
    6   7   8              6   7   8

OPERATOR            17


    1   2   3              1   2   3
    4   5   0    :=        4   5   8
    6   7   8              6   7   0

OPERATOR            18


    1   2   3              1   2   3
    4   5   8    :=        4   5   0
    6   7   0              6   7   8

OPERATOR            19


    1   2   3              1   2   0
    4   5   0    :=        4   5   3
    6   7   8              6   7   8

OPERATOR            20


    1   2   0              1   2   3
    4   5   3    :=        4   5   0
    6   7   8              6   7   8

OPERATOR            21


    1   2   3              1   2   3
    4   7   5    :=        4   7   5
    6   0   8              0   6   8

OPERATOR            22


    1   2   3              1   2   3
    4   7   5    :=        4   7   5
    0   6   8              6   0   8

OPERATOR            23


    1   2   3              1   2   3
    4   7   5    :=        4   7   5
    6   0   8              6   8   0

OPERATOR            24


    1   2   3              1   2   3
```

```
   4    7    5      :=       4    7    5
   6    8    0               6    0    8

-------------------------------
 PROVE THAT

     2     8     3
     1     6     4
     7     0     5
IS EQUIVALENT TO

     1     2     3
     8     0     4
     7     6     5
-------------------------------

SOLUTION TIME(SECS):                 13

PENETRANCE      7.1429,-01

EFFECTIVE BRANCHING FACTOR:               1.0915,+00

                                                          APPLY OP  0
     2     8     3
     1     6     4
     7     0     5
                                                          APPLY OP  4
     2     8     3
     1     0     4
     7     6     5
                                                          APPLY OP  7
     2     0     3
     1     8     4
     7     6     5
                                                          APPLY OP 13
     0     2     3
     1     8     4
     7     6     5
                                                          APPLY OP 10
     1     2     3
     0     8     4
     7     6     5
                                                          APPLY OP  2
     1     2     3
     8     0     4
     7     6     5
```

```
@HDG EIGHT PUZZLES TWO
@XQT
BEGIN EXECUTION. NU ALGOL LIBRARY LEVEL 6.2, FEB. 15, 1974

-------------------------------
PROVE THAT


    2   8   1
    4   0   3
    7   6   5

IS EQUIVALENT TO


    1   2   3
    8   0   4
    7   6   5

-------------------------------
SOLUTION TIME(SECS):                      58

PENETRANCE:    2.1569,-01

EFFECTIVE BRANCHING FACTOR:                  1.2481,+00

                                                        APPLY OP  0
    2   8   1
    4   0   3
    7   6   5

                                                        APPLY OP  1
    2   8   1
    0   4   3
    7   6   5

                                                        APPLY OP  9
    0   8   1
    2   4   3
    7   6   5

                                                        APPLY OP 14
    8   0   1
    2   4   3
    7   6   5

                                                        APPLY OP 15
    8   1   0
    2   4   3
    7   6   5

                                                        APPLY OP 20
    8   1   3
    2   4   0
```

```
7    6    5

                                             APPLY OP  6

8    1    3
2    0    4
7    6    5

                                             APPLY OP  1

8    1    3
0    2    4
7    6    5

                                             APPLY OP  9

0    1    3
8    2    4
7    6    5

                                             APPLY OP 14

1    0    3
8    2    4
7    6    5

                                             APPLY OP  8

1    2    3
8    0    4
7    6    5
```

```
@HDG BOOLEAN ALGEGRA PROBLEM
@XQT
BEGIN EXECUTION. NU ALGOL LIBRARY LEVEL 6.2, FEB. 15, 1974

OPERATOR                  1

V1.WEDGE.V1
V1

OPERATOR                  2

V1
V1.WEDGE.V1

OPERATOR                  3

V1.VEE.V1
V1

OPERATOR                  4

V1
V1.VEE.V1

OPERATOR                  5

V1.WEDGE.V2
V2.WEDGE.V1

OPERATOR                  6

V2.VEE.V1
V1.VEE.V2

OPERATOR                  7

V1.WEDGE.(V2.WEDGE.V3)
V1.WEDGE.V2.WEDGE.V3

OPERATOR                  8

V1.VEE.(V2.VEE.V3)
V1.VEE.V2.VEE.V3

OPERATOR                  9

V1.WEDGE.(V1.VEE.V2)
V1

OPERATOR                 10

V1.VEE.(V1.WEDGE.V2)
V1

OPERATOR                 11
```

```
V1.WEDGE.(V2.VEE.V3)
V1.WEDGE.V2.VEE.(V1.WEDGE.V3)

OPERATOR                    12

V1.VEE.(V2.WEDGE.V3)
V1.VEE.V2.WEDGE.(V1.VEE.V3)

-------------------------------
 PROVE THAT

X.WEDGE.(Y.VEE.X.WEDGE.Z)

IS EQUIVALENT TO

X.WEDGE.Y.VEE.(X.WEDGE.Z)

-------------------------------

SOLUTION TIME(SECS):              38

PENETRANCE     1.3953,-01

EFFECTIVE BRANCHING FACTOR:            1.2674,+00

                                                  APPLY OP  0
X.WEDGE.(Y.VEE.X.WEDGE.Z)

                                                  APPLY OP 12
X.WEDGE.(Y.VEE.X.WEDGE.Y.VEE.Z)

                                                  APPLY OP  7
X.WEDGE.Y.VEE.X.WEDGE.(Y.VEE.Z)

                                                  APPLY OP  6
X.WEDGE.X.VEE.Y.WEDGE.(Y.VEE.Z)

                                                  APPLY OP  9
X.WEDGE.(Y.VEE.Z)

                                                  APPLY OP 11
X.WEDGE.Y.VEE.(X.WEDGE.Z)
```

```
@HDG PROPOSITIONAL CALCULUS PROBLEMS
@XQT
BEGIN EXECUTION. NU ALGOL LIBRARY LEVEL 6.2. FEB. 15, 1974

OPERATOR                    1

V1.AND.V2
V2.AND.V1

OPERATOR                    2

V1.AND.(V2.AND.V3)
V1.AND.V2.AND.V3

OPERATOR                    3

V1.AND.(V1.IM.V2)
V2

OPERATOR                    4

.NOT.V1.AND.(V2.IM.V1)
.NOT.V2.

OPERATOR                    5

.NOT..NOT.V1
V1

OPERATOR                    6

V1
.NOT..NOT.V1

OPERATOR                    7

.NOT.V1.IM..NOT.V2
V2.IM.V1


------------------------------
 PROVE THAT

A.IM..NOT.B.AND.B

IS EQUIVALENT TO

.NOT.A

------------------------------

SOLUTION TIME(SECS):                   1

PENETRANCE     3.3333,-01
```

```
EFFECTIVE BRANCHING FACTOR:             1.2973.+00

                                                            APPLY OP  0
A.IM..NOT.B.AND.B

                                                            APPLY OP  1
B.AND.(A.IM..NOT.B)

                                                            APPLY OP  6
.NOT..NOT.B.AND.(A.IM..NOT.B)

                                                            APPLY OP  4
.NOT.A

--------------------------------
 PROVE THAT
B.AND.(.NOT.A.IM..NOT.B)
IS EQUIVALENT TO
A
--------------------------------
SOLUTION TIME(SECS):                     0
PENETRANCE     1.0000.+00
EFFECTIVE BRANCHING FACTOR:  1.0

                                                            APPLY OP  0
B.AND.(.NOT.A.IM..NOT.B)

                                                            APPLY OP  7
B.AND.(B.IM.A)

                                                            APPLY OP  3
A

--------------------------------
 PROVE THAT
.NOT.B.AND.A.IM.B.AND.(.NOT.A.IM.C)
IS EQUIVALENT TO
C
--------------------------------
SOLUTION TIME(SECS):                     2
PENETRANCE     5.0000.-01
```

```
EFFECTIVE BRANCHING FACTOR:                1.2064.+00

                                                         APPLY OP  0
.NOT.B.AND.A.IM.B.AND.(.NOT.A.IM.C)

                                                         APPLY OP  4
.NOT.A.AND.(.NOT.A.IM.C)

                                                         APPLY OP  3
C

--------------------------------
 PROVE THAT
.NOT.C.AND.B.IM.C.AND.(.NOT.B.IM..NOT..NOT.A)
IS EQUIVALENT TO
A
--------------------------------
SOLUTION TIME(SECS):                  1
PENETRANCE     1.0000.+00
EFFECTIVE BRANCHING FACTOR:  1.0

                                                         APPLY OP  0
.NOT.C.AND.B.IM.C.AND.(.NOT.B.IM..NOT..NOT.A)

                                                         APPLY OP  5
.NOT.C.AND.B.IM.C.AND.(.NOT.B.IM.A)

                                                         APPLY OP 10
A

--------------------------------
 PROVE THAT
A.IM..NOT.B.AND.B.AND.(.NOT.A.IM.C.AND.D)
IS EQUIVALENT TO
C.AND.D
--------------------------------
SOLUTION TIME(SECS):                  8
PENETRANCE     3.3333.-01
EFFECTIVE BRANCHING FACTOR:                1.1409.+00

                                                         APPLY OP  0
```

```
A.IM..NOT.B.AND.B.AND.(.NOT.A.IM.C.AND.D)
                                                          APPLY OP  1
B.AND.A.IM..NOT.B.AND.(.NOT.A.IM.C.AND.D)
                                                          APPLY OP  6
.NOT..NOT.B.AND.A.IM..NOT.B.AND.(.NOT.A.IM.C.AND.D)
                                                          APPLY OP 10
C.AND.D
```

```
@HDG ELEMENTARY ALGEBRA PROBLEMS
@XQT
BEGIN EXECUTION. NU ALGOL LIBRARY LEVEL 6.2, FEB. 15, 1974

OPERATOR                    1

V1+V2
V2+V1

OPERATOR                    2

V1+(V2+V3)
V1+V2+V3

OPERATOR                    3

V1+V2-V2
V1

OPERATOR                    4

V1
V1+V2-V2

OPERATOR                    5

V1-V2+V3
V1+V2-V3

OPERATOR                    6

V1+V2-V3
V1-V3+V2


--------------------------------
 PROVE THAT

X+Y+Z

IS EQUIVALENT TO

X+(Y+Z)

--------------------------------

SOLUTION TIME(SECS):                      6

PENETRANCE      4.1667,-01

EFFECTIVE BRANCHING FACTOR:                    1.1697,+00

                                                                  APPLY OP   0
X+Y+Z
```

```
                                                    APPLY OP  1
Z+(X+Y)
                                                    APPLY OP  1
Z+(Y+X)
                                                    APPLY OP  2
Z+Y+X
                                                    APPLY OP  1
X+(Z+Y)
                                                    APPLY OP  1
X+(Y+Z)

------------------------------
 PROVE THAT
X-Y+Y
IS EQUIVALENT TO
X
------------------------------
SOLUTION TIME(SECS):              0
PENETRANCE     1.0000,+00
EFFECTIVE BRANCHING FACTOR:  1.0
                                                    APPLY OP  0
X-Y+Y
                                                    APPLY OP  5
X+Y-Y
                                                    APPLY OP  3
X

------------------------------
 PROVE THAT
X
IS EQUIVALENT TO
X-Y+Y
------------------------------
SOLUTION TIME(SECS):              1
PENETRANCE     6.0000,-01
```

EFFECTIVE BRANCHING FACTOR: 1.1370,+00

APPLY OP 0

X

APPLY OP 4

X+V2-V2

APPLY OP 6

X-V2+V2

--------------------------------

PROVE THAT

X+(Y-Z)

IS EQUIVALENT TO

X+Y-Z

--------------------------------

SOLUTION TIME(SECS): 7

PENETRANCE 2.1429,-01

EFFECTIVE BRANCHING FACTOR: 1.4708,+00

APPLY OP 0

X+(Y-Z)

APPLY OP 4

X+Y-Z+V2-V2

APPLY OP 7

X+Y-Z+V2-V2

APPLY OP 8

X+Y-Z

--------------------------------

PROVE THAT

X-Y+Z

IS EQUIVALENT TO

X+(Z-Y)

--------------------------------

SOLUTION TIME(SECS): 8

PENETRANCE 2.5000,-01

```
EFFECTIVE BRANCHING FACTOR:              1.2077++00

                                                        APPLY OP  0
X-Y+Z

                                                        APPLY OP  1
Z+(X-Y)

                                                        APPLY OP 10
Z+X-Y

                                                        APPLY OP  6
Z-Y+X

                                                        APPLY OP  1
X+(Z-Y)

EXECUTION TIME      29.317 SECONDS
```

```
@HDG ELEMENTARY ALGEBRA PROBLEMS
@XQT
BEGIN EXECUTION. NU ALGOL LIBRARY LEVEL 6.2, FEB. 15, 1974

-------------------------------
 PROVE THAT

X-Z-(Y-Z)

IS EQUIVALENT TO

X-Y

-------------------------------

SOLUTION TIME(SECS):                    274

PENETRANCE     6.7431,-02

EFFECTIVE BRANCHING FACTOR:                 1.3384,+00


-------------------------------
 PROVE THAT

X+Z-(Y+Z)

IS EQUIVALENT TO

X-Y

-------------------------------

SOLUTION TIME(SECS):                    204

PENETRANCE     7.3172,-02

EFFECTIVE BRANCHING FACTOR:                 1.3726,+00

-------------------------------
 PROVE THAT

X+(Y-Z)

IS EQUIVALENT TO

X-Z+Y

-------------------------------

SOLUTION TIME(SECS):                    1

PENETRANCE     6.6667,-01
```

```
EFFECTIVE BRANCHING FACTOR:              1.0569,+00

-----------------------------
 PROVE THAT

X-Y-Z

IS EQUIVALENT TO

X-Z-Y

-----------------------------

SOLUTION TIME(SECS):                 17

PENETRANCE     1.8181,-01

EFFECTIVE BRANCHING FACTOR:              1.5376,+00

-----------------------------
 PROVE THAT

X+Y-Z

IS EQUIVALENT TO

X+(Y-Z)

-----------------------------

SOLUTION TIME(SECS):                 1

PENETRANCE     1.0000,+00

EFFECTIVE BRANCHING FACTOR:  1.0

-----------------------------
 PROVE THAT

X-(Y+Z)

IS EQUIVALENT TO

X-Y-Z

-----------------------------

SOLUTION TIME(SECS):                 21

PENETRANCE     1.5384,-01

EFFECTIVE BRANCHING FACTOR:              1.6424,+00
```

```
--------------------------------
 PROVE THAT

X+(Y-Z)

IS EQUIVALENT TO

X-(Z-Y)

--------------------------------

SOLUTION TIME(SECS):                    13

PENETRANCE     2.6086,-01

EFFECTIVE BRANCHING FACTOR:                 1.4097,+00

--------------------------------
 PROVE THAT

X-Y+Z

IS EQUIVALENT TO

X-(Y-Z)

--------------------------------

SOLUTION TIME(SECS):                    3

PENETRANCE     3.3333,-01

EFFECTIVE BRANCHING FACTOR:                 1.5613,+00
```

```
@HDG FATHER AND SONS PROBLEM
@XQT
BEGIN EXECUTION. NU ALGOL LIBRARY LEVEL 6.2, FEB. 15, 1974

OPERATOR                 1

V1.LEFTBANK.2.AND..BOAT..LEFT.
V1.LEFTBANK.1.AND..BOAT..RIGHT.

OPERATOR                 2

V1.LEFTBANK.1.AND..BOAT..LEFT.
V1.LEFTBANK.0.AND..BOAT..RIGHT.

OPERATOR                 3

V1.LEFTBANK.1.AND..BOAT..RIGHT.
V1.LEFTBANK.2.AND..BOAT..LEFT.

OPERATOR                 4

V1.LEFTBANK.0.AND..BOAT..RIGHT.
V1.LEFTBANK.1.AND..BOAT..LEFT.

OPERATOR                 5

V1.LEFTBANK.2.AND..BOAT..LEFT.
V1.LEFTBANK.0.AND..BOAT..RIGHT.

OPERATOR                 6

V1.LEFTBANK.0.AND..BOAT..RIGHT.
V1.LEFTBANK.2.AND..BOAT..LEFT.

OPERATOR                 7

1.LEFTBANK.V1.AND..BOAT..LEFT.
0.LEFTBANK.V1.AND..BOAT..RIGHT.

OPERATOR                 8

0.LEFTBANK.V1.AND..BOAT..RIGHT.
1.LEFTBANK.V1.AND..BOAT..LEFT.


------------------------------
 PROVE THAT

1.LEFTBANK.2.AND..BOAT..LEFT.

IS EQUIVALENT TO

0.LEFTBANK.0.AND..BOAT..RIGHT.

------------------------------
```

```
SOLUTION TIME(SECS):                    7

PENETRANCE     2.0000,-01

EFFECTIVE BRANCHING FACTOR:                 1.1440,+00

                                                              APPLY OP  0
1.LEFTBANK.2.AND..BOAT..LEFT.

                                                              APPLY OP  5
1.LEFTBANK.0.AND..BOAT..RIGHT.

                                                              APPLY OP  4
1.LEFTBANK.1.AND..BOAT..LEFT.

                                                              APPLY OP  7
0.LEFTBANK.1.AND..BOAT..RIGHT.

                                                              APPLY OP  3
0.LEFTBANK.2.AND..BOAT..LEFT.

                                                              APPLY OP  5
0.LEFTBANK.0.AND..BOAT..RIGHT.

EXECUTION TIME        9.743 SECONDS
@BRKPT PRINT$
```

```
@XQT
BEGIN EXECUTION, NU ALGOL LIBRARY LEVEL 6.2, FEB. 15, 1974

OPERATOR                  1

V1.SAYS.V2
V1.IS..TTLR..EQ.V2

OPERATOR                  2

V1.IS..LIAR.
.NOT.(V1.IS..TTLR.)

OPERATOR                  3

.NOT..EAST.
.WEST.

OPERATOR                  4

.NOT..WEST.
.EAST.

OPERATOR                  5

V1.EQ.V2.AND..NOT.V2
.DATA..IM..NOT.V1

OPERATOR                  6

V1.EQ.V2
V2.EQ.V1

OPERATOR                  7

V1.EQ.(V2.EQ.V3)
V1.EQ.V2.EQ.V3

OPERATOR                  8

V1.EQ..NOT.V2
.NOT.(V1.EQ.V2)


 PROVE THAT

.AL..SAYS..ED..SAYS..EAST..AND.(.AL..SAYS..ED..IS..LIAR.)

IS EQUIVALENT TO

.DATA..IM..DIRN.


SOLUTION TIME(SECS):            32

PENETRANCE    4.5000,-01
```

```
EFFECTIVE BRANCHING FACTOR:          1.1117.+00

                                                                          APPLY OP
.AL..SAYS..ED..SAYS..EAST..AND.(.AL..SAYS..ED..IS..LIAR.)

                                                                          APPLY OP
.AL..SAYS..ED..SAYS..EAST..AND.(.AL..IS..TTLR..EQ..ED..IS..LIAR.)

                                                                          APPLY OP
.AL..SAYS..ED..SAYS..EAST..AND.(.AL..IS..TTLR..EQ..NOT..ED..IS..TTLR.)

                                                                          APPLY OP
.AL..SAYS..ED..SAYS..EAST..AND..NOT..AL..IS..TTLR..EQ..ED..IS..TTLR.

                                                                          APPLY OP
.AL..IS..TTLR..EQ..ED..SAYS..EAST..AND..NOT..AL..IS..TTLR..EQ..ED..IS..TTLR.

                                                                          APPLY OP
.AL..IS..TTLR..EQ..ED..IS..TTLR..EQ..EAST..AND..NOT..AL..IS..TTLR..EQ..ED..IS..TTLR.

                                                                          APPLY OP
.AL..IS..TTLR..EQ..ED..IS..TTLR..EQ..EAST..AND..NOT..AL..IS..TTLR..EQ..ED..IS..TTLR.

                                                                          APPLY OP
.EAST..EQ..AL..IS..TTLR..EQ..ED..IS..TTLR..AND..NOT..AL..IS..TTLR..EQ..ED..IS..TTLR.

                                                                          APPLY OP
.DATA..IN..NOT..EAST.

                                                                          APPLY OP
.DATA..IN..WEST.

EXECUTION TIME     34.301 SECONDS
SBRKIT PRINTS
```

```
@HDG MONKEY AND BANANAS PROBLEM
@XQT
BEGIN EXECUTION. NU ALGOL LIBRARY LEVEL 6.2, FEB. 15, 1974

OPERATOR                    1

.NOT..ONBOX..AND.(.MON..AT.A)
.NOT..ONBOX..AND.(.MON..AT.B)

OPERATOR                    2

.NOT..ONBOX..AND..MON..AT.B.AND.(.BOX..AT.B)
.NOT..ONBOX..AND..MON..AT.C.AND.(.BOX..AT.C)

OPERATOR                    3

.NOT..ONBOX..AND..MON..AT.V1.AND.(.BOX..AT.V1)
.ONBOX..AND.(.BOX..AT.V1)

OPERATOR                    4

.ONBOX..AND..BOX..AT.C.AND..NOT..HB.
.HB.


--------------------------------
 PROVE THAT

.NOT..ONBOX..AND..MON..AT.A.AND..BOX..AT.B.AND..NOT..HB.

IS EQUIVALENT TO

.HB.

--------------------------------

SOLUTION TIME(SECS):                1

PENETRANCE     1.0000+00

EFFECTIVE BRANCHING FACTOR:  1.0

                                                         APPLY OP  0
.NOT..ONBOX..AND..MON..AT.A.AND..BOX..AT.B.AND..NOT..HB.

                                                         APPLY OP  1
.NOT..ONBOX..AND..MON..AT.B.AND..BOX..AT.B.AND..NOT..HB.

                                                         APPLY OP  2
.NOT..ONBOX..AND..MON..AT.C.AND..BOX..AT.C.AND..NOT..HB.

                                                         APPLY OP  3
.ONBOX..AND..BOX..AT.C.AND..NOT..HB.

                                                         APPLY OP  4
.HB.
```

# APPENDIX B

## THE SDPS SYSTEM

MULTIPLE.SOPS.F2

```
        BEGIN
        COMMENT


            ••••••••••••••••••••••••••••••••••••••••••••••••••••••••
            ••                                                    ••
            ••      THE SOPS PROBLEM SOLVING SYSTEM               ••
            ••                                                    ••
            ••••••••••••••••••••••••••••••••••••••••••••••••••••••••


        *
           INTEGER MAXELT,NEXTELT,TOPGL,MAXGL,NEXTGOAL,MAXGOALS,FACTOR,
           MAXNODES,NEXTNODE,MAXOPS,FREEOPS,LASTOP,MAXSTRING,MAXTIME,
          MAXSUBGOALS,NEXTSUBOP,MAXSUBOPS,MAXRULES,MAXSYMBOLS,MAXDIFFS,TOTRULE,
          MAXTABSIZE,SYMTOT,MAXSTAX,LENGTHBIAS $
           REAL  EVALOPS,EVALDEPTH,EVALSUB,ERR,DEPTHBIAS1,DEPTHBIAS2,DEPTHBIAS3,
           DEPTHBIAS4,DIFFBIAS1,DIFFBIAS2,DIFFBIAS3,DIFFBIAS4,COMPBIAS,SPECBIAS,
            DIFFACTOR1,DIFFACTOR2,RCFACTOR1,RCFACTOR2,RCFACTOR3,RCFACTOR4,
            RCFACTOR5 $


           LIST MAXVALUES1(MAXGOALS,MAXGL,MAXSUBOPS,MAXELT,MAXTIME,MAXOPS,
                MAXNODES,MAXSTRING) $
           LIST MAXVALUES2(MAXSUBGOALS,MAXSYMBOLS,MAXDIFFS,TOTRULE,MAXTABSIZE,
                SYMTOT,MAXSTAX,MAXRULES) $
           LIST EVALTYPE(EVALOPS,EVALDEPTH,EVALSUB) $
           LIST DEPTHTYPE(DEPTHBIAS1,DEPTHBIAS2,DEPTHBIAS3,DEPTHBIAS4) $
           LIST DIFFTYPE(DIFFBIAS1,DIFFBIAS2,DIFFBIAS3,DIFFBIAS4,DIFFACTOR1,
                DIFFACTOR2) $
           LIST RCTYPE(RCFACTOR1,RCFACTOR2,RCFACTOR3,RCFACTOR4,RCFACTOR5) $
           FORMAT FMAX(A,8I5) $
           FORMAT FEVAL(A,3R6.3) $
           FORMAT FDEPTH(A,4R6.3) $
           FORMAT FDIFF(A,6R6.3) $
           FORMAT FRC(A,5R6.3) $

         PROCEDURE MAIN1 $


        COMMENT MAIN1 ENCLOSES THE ENTIRE SUITE OF PROCEDURES, IT ALSO DEFINES
         THE GLOBAL ARRAYS AND VARIABLES $

          BEGIN
           INTEGER  ARRAY     SYMTAB(1:MAXSYMBOLS,1:5),
           DIFFS(1:MAXDIFFS,1:2),DIFFSET(0:10),RULE(1:TOTRULE,1:5),
           RULESH(1:MAXRULES,1:2),RULESL(1:MAXRULES,1:2),
           OPLIST(1:MAXDIFFS,1:3),ELT(1:MAXELT,1:5),BUFFER(1:MAXSTRING),
           GOALIST(1:MAXGL,1:2),GOALS(1:MAXGOALS),NODE(1:MAXNODES,1:10),
           OPER(1:MAXOPS,1:4),STRA(1:MAXSTRING,1:5),STRB(1:MAXSTRING,1:5),
           TERMTAB(1:MAXTABSIZE,1:4),VARTAB(1:MAXTABSIZE,1:4),
           SUBOPLIST(1:MAXSUBOPS),TERMTABENTRY(1:MAXRULES) $
           STRING SYMVALUE(SYMTOT) $
```

```
        REAL ARRAY RCOMB(1:MAXRULES),OPVALUE(1:MAXDIFFS),NODEVAL(1:MAXNODES),
        OPERVAL(1:MAXOPS) $
        INTEGER I,J,M,N,C1,C2,P1,L1,N2,L2,P2,C3,DIFFNUM,PMAIN,RULEND,
        OPNUM,OP1,P,LENGA,LENGB,OPDLEVEL,TTBPNT,VATPNT  $

    COMMENT ASSEMBLER PROCEDURES ARE USED TO PACK AND UNPACK DATA TO PARTIAL
      WORDS $

        EXTERNAL LIBRARY PROCEDURE PACK6(A,B) $
        INTEGER ARRAY A $
        INTEGER B $ $
        EXTERNAL LIBRARY PROCEDURE UNPACK6(B,A) $
        VALUE A $
        INTEGER ARRAY B $
        INTEGER A $ $
        EXTERNAL LIBRARY PROCEDURE PACK2(A,B,C) $
        VALUE A,B $
        INTEGER A,B,C $ $
        EXTERNAL LIBRARY PROCEDURE UNPACK2(A,B,C) $
        VALUE A $
        INTEGER A,B,C $$
        EXTERNAL LIBRARY PROCEDURE PACK3(A,B,C,D) $
        VALUE A,B,C $
        INTEGER A,B,C,D $ $
        EXTERNAL LIBRARY PROCEDURE UNPACK3(A,B,C,D) $
        VALUE A $
        INTEGER A,B,C,D $ $



        BOOLEAN PROCEDURE NAMELT(BUFFER,NAME) $
        INTEGER ARRAY BUFFER $
        INTEGER NAME $

    COMMENT PROCEDURE DETERMINES WHETHER A STRUCTURE HAS BEEN FILED IN THE
     TREE. IF NOT IT IS FILED AND THE NAME RETURNED WITH THE PROCEDURE
     VALUE SET TO FALSE $

        BEGIN
        INTEGER P1,P2,C1,BUFFLENG,NEWLINK,BACKPNT $
        BOOLEAN OLDELT,NEWIN $



         PROCEDURE SEARCHNAMETREE(TREEPNT,BUFFPNT) $
         INTEGER  TREEPNT,BUFFPNT $

    COMMENT PROCEDURE IS USED FOR RECURSIVE SEARCH OF THE TREE $

        BEGIN
         IF BUFFER(BUFFPNT,1) EQL ELT(TREEPNT,1) THEN
         BEGIN

    COMMENT IF ELEMENTS ARE EQUAL THEN IF THE BUFFER IS FULLY SEARCHED THE
     STRUCTURE IS NOT NEW $

          IF BUFFPNT EQL BUFFLENG THEN
```

```
          BEGIN
           NEWIN = FALSE $
           NAME = TREEPNT $
           IF ELT(TREEPNT,2) EQL 0 THEN
            BEGIN
            OLDELT = FALSE $
            ELT(TREEPNT,2) = TREEPNT $
            END
           ELSE OLDELT = TRUE $
          END
         ELSE BEGIN

     COMMENT IF THE RIGHT BRANCH IS EMPTY THE OBJECT IS NEW OTHERWISE SEARCH
      THE TREE ROOTED AT THE RIGHT BRANCH $

          IF ELT(TREEPNT,4) EQL 0 THEN
           BEGIN
            OLDELT = FALSE $
            BACKPNT = BUFFPNT + 1 $
            NEWLINK = TREEPNT $
            NEWIN = TRUE $
           END
          ELSE SEARCHNAMETREE(ELT(TREEPNT,4),BUFFPNT + 1) $
         END
        END

     COMMENT IF ELEMENTS ARE NOT EQUAL AND THE LEFT BRANCH IS NON-EMPTY ,
      SEARCH THE TREE ROOTED AT THE LEFT BRANCH OTHERWISE THE OBJECT IS NEW

       ELSE
         IF ELT(TREEPNT,3) NEQ 0 THEN
          SEARCHNAMETREE(ELT(TREEPNT,3),BUFFPNT)
          ELSE BEGIN
          OLDELT = FALSE $
          ELT(TREEPNT,3) = NEWLINK = NEXTELT $
          IF NEXTELT GEQ MAXELT THEN ERRORWRITE(1)
          ELSE NEXTELT = NEXTELT + 1$
          ELT(NEWLINK,1) = BUFFER(BUFFPNT,1) $
          ELT(NEWLINK,5) = TREEPNT $
          BACKPNT = BUFFPNT + 1$
          IF BACKPNT GTR BUFFLENG THEN
           BEGIN
            NEWIN = FALSE $
            ELT(NEWLINK,2) = NAME = NEWLINK $
           END
          ELSE NEWIN = TRUE $
          END$
         END$


        BEGIN
         P1 = P2 = 1 $
         BUFFLENG = BUFFER(P1,4) $
         SEARCHNAMETREE(P1,P2) $

     COMMENT IF AN OLD STRUCTURE THE PROCEDURE IS TRUE OTHERWISE INSERT THE
      REST OF THE STRUCTURE INTO THE CANONICAL TREES
```

```
    IF OLDELT THEN NAMELT = TRUE
    ELSE BEGIN
     NAMELT = FALSE $
     IF NEWIN THEN BEGIN
      FOR CI = BACKPNT STEP 1 UNTIL BUFFLENG DO
       BEGIN
       ELT(NEWLINK,4) = NEXTELT $
       ELT(NEXTELT,5) = NEWLINK $
       ELT(NEXTELT,1) = BUFFER(CI,1) $
       NEWLINK = NEXTELT $
       IF NEXTELT GEQ MAXELT THEN ERRORWRITE(1)
        ELSE NEXTELT = NEXTELT + 1 $
      END$
      ELT(NEWLINK,2) = NAME = NEWLINK $
     END$
     END$
    END$
   END$



     PROCEDURE RETRIEVELT(NAME,BUFFER,BUFFLENG) $
     INTEGER ARRAY BUFFER $
     INTEGER NAME,BUFFLENG $

 COMMENT PROCEDURE RETURNS AN OBJECT STRUCTURE FROM THE CANONICAL TREE
  BY BUILDING A BUFFER OF ELEMENTS IN NODE NUMBER ORDER.
   IT BACKS UP FROM THE NAME OF THE STRUCTURE AND OUTPUTS EACH VALUE
    REACHED BY A RIGHT BRANCH $

    BEGIN
    INTEGER P1,P2,LAST $
    INTEGER ARRAY DUMMY(1:100) $
    IF ELT(NAME,2) EQL 0 THEN ERRORWRITE(2) $
    P1 = 1 $
    P2 = NAME $
    DUMMY(1) = ELT(NAME,1) $
L1:
    IF P2 NEQ 1 THEN BEGIN
     LAST = P2 $
    P2 = ELT(P2,5) $
    IF ELT(P2,4) EQL LAST THEN
     BEGIN
      P1 = P1 + 1 $
      DUMMY(P1) = ELT(P2,1) $
     END$
    GO TO L1 $
    END$
    BUFFLENG = P1 $
    LAST = 1 $
    FOR P2 = P1 STEP -1 UNTIL 1 DO
     BEGIN
     BUFFER(LAST,1) = DUMMY(P2) $
     LAST = LAST + 1 $
     END$
   END$
```

```
    PROCEDURE PACKELT(ELT,OP,A,P,N) $
    INTEGER ARRAY A $
    INTEGER ELT,OP,P,N $

COMMENT PROCEDURE PACKS THE POSITION OF AN 'ESSENTIAL' ELEMENT FOR
 AN OPERATOR $

   BEGIN
   INTEGER ARRAY STACK1(1:6),STACK2(1:MAXSTAX) $
    INTEGER I,J,C1,C2,C3 $
    I = ELT + P $
    J = MOD(OP,FACTOR) + P  $
    C1 = 0 $
    IF ELT LSS 0 THEN NODE(N,6) = ELT
     ELSE BEGIN
     FOR C1=C1+1 WHILE I NEQ J DO
      BEGIN
      STACK2(C1) = A(I,2) $
      I = A(I,3) $
      END$
      C1 = C1 -1 $
     IF C1 GTR 5 THEN ERRORWRITE(10) $
     C3 = 1$
     FOR C2=(C1,-1,1) DO
      BEGIN
      STACK1(C3) = STACK2(C2) $
      C3 = C3+ 1 $
      END$
     STACK1(6) = C1 $
     PACK6(STACK1,NODE(N,6)) $
    END$
    END$



    PROCEDURE PACKSUBOP(OP,STR,P,NEXT) $
    INTEGER ARRAY STR $
    INTEGER OP,P,NEXT $

COMMENT PROCEDURE PACKS THE POSITION AT WHICH AN OP IS TO BE APPLIED $

   BEGIN
    INTEGER I,J,N,C1,C2 $
    INTEGER ARRAY STACK1(1:6),STACK2(1:MAXSTAX) $
    I = OP//FACTOR $
    J = MOD(OP,FACTOR) + P$
    N = C1 = 0 $

COMMENT IF DEPTH OF OP LSS 2 THEN INSERT TO NODE DIRECTLY OTHERWISE
   STACK POSITIONS IN SUBOPLIST AND SET POINTERS AT NODE $

    IF STR(J,3) EQL 0 OR STR(STR(J,3),3) EQL 0 THEN BEGIN
     N = 0 $ C2 = STR(J,2) $ END
   ELSE BEGIN
```

```
      C2 = NEXTSUBOP + 1 $
      FOR N = N+1 WHILE STR(J,3) NEQ 0 DO
       BEGIN
        STACK2(N) = STR(J,2) $
        J = STR(J,3) $
       END$
       N = N - 1 $
       J = N $
       FOR NEXTSUBOP = NEXTSUBOP + 1 WHILE J GTR 0 DO
        BEGIN
        FOR C1 =(1,1,6) DO
         IF J GTR 0 THEN
          BEGIN
          STACK1(C1) = STACK2(J) $
          J = J-1 $
          END
         ELSE STACK1(C1) = 0 $
         PACK6(STACK1,SUBOPLIST(NEXTSUBOP)) $
       END$
      NEXTSUBOP = NEXTSUBOP - 1 $
      END$
      PACK3(I,C2,N,NODE(NEXT,7)) $
      END$



      PROCEDURE UNPACKOP(OP,STR,P1,NN) $
      INTEGER ARRAY STR $
      INTEGER OP,P1,NN $

   COMMENT PROCEDURE UNPACKS THE POSITION AT WHICH AN OPERATOR IS TO BE
    APPLIED IN STRUCTURE STR, USED FOR RESTRICTION OF OPERATORS $

      BEGIN
      INTEGER I,J,N,PNT,C1,C2,C3 $
      INTEGER ARRAY STACK1(1:6) $
      UNPACK3(NODE(NN,7),I,PNT,N) $
      J = P1 $

   COMMENT IF DEPTH LSS 2 THEN SELECT POSITION DIRECTLY OTHERWISE UNPACK
    THE STACK OF POINTERS $

      IF N EQL 0 THEN
       BEGIN
       IF PNT LSS 2 THEN J = J + PNT
      ELSE BEGIN
      J = J + 1 $
      FOR C3 =(2,1,PNT)DO J = STR(J,4) + 1 $
      END
       END
      ELSE BEGIN
       PNT = PNT - 1 $
       FOR PNT = PNT + 1 WHILE N GTR 0 DO
        BEGIN
        IF N GTR 6 THEN C1 = 6
          ELSE C1 = N $
         UNPACK6(STACK1,SUBOPLIST(PNT)) $
```

```
          FOR C2=(1,1,C1) DO
           BEGIN
            J = J + 1 $
            FOR C3 = (2,1,STACKI(C2)) DO
             J = STR(J,4) + I $
           END$
          N = N-6 $
          END$
         END$
        OP = I * FACTOR + J - P1 $
       END$


       BOOLEAN PROCEDURE BINSEARCH(G,N) $
       INTEGER G,N $

   COMMENT PROCEDURE PERFORMS A BINARY SEARCH ON THE LIST OF FIRST ELEMEN
    OF AN OBJECT/GOAL PAIR AND RETURNS THE ELEMENT POSITION PLUS THE
     VALUE TRUE IF IT EXISTS $

       BEGIN
        INTEGER I,UPPER,LOWER $
        LOWER = 1 $
        UPPER = TOPGL + 1 $
   S1:
       I = (UPPER + LOWER)//2 $
        IF GOALIST(I,1) EQL G THEN
        BEGIN
         N = I $
         BINSEARCH = TRUE $
         GO TO S2 $
        END$
        IF GOALIST(I,1) LSS G THEN
        BEGIN
         IF LOWER EQL I THEN GO TO S3 $
         LOWER = I $
         GO TO S1 $
        END
        ELSE BEGIN
         IF UPPER EQL I THEN GO TO S3  $
        UPPER = I $
        GO TO S1 $
       END$
   S3:
       BINSEARCH = FALSE $
   S2:
      END$


       BOOLEAN PROCEDURE TESTGOAL(G1,G2,N) $
       INTEGER G1,G2,N $

   COMMENT PROCEDURE DETERMINES WHETHER AN OBJECT/GOAL PAIR HAS OCCURRED
```

```
      PREVIOUSLY BY SEARCHING THE LIST OF STORED PAIRS $

       BEGIN
        INTEGER I,CI,NEXT,VAL,DEP $
         IF BINSEARCH(GI,I) THEN
         BEGIN
        CI = GOALIST(I,2) $
    SI:
      UNPACK3(GOALS(CI),VAL,NEXT,DEP) $
       IF VAL NEQ G2 THEN
        BEGIN
        IF NEXT NEQ O THEN BEGIN
         CI = NEXT $
         GO TO SI $
         END
        ELSE BEGIN
         NEXTGOAL  = NEXTGOAL + 1 $
         IF NEXTGOAL GTR MAXGOALS THEN ERRORWRITE(4) $
        PACK3(VAL,NEXTGOAL,DEP,GOALS(CI)) $
        PACK3(G2,0,N,GOALS(NEXTGOAL)) $
         TESTGOAL = FALSE $
        END
       END
       ELSE BEGIN TESTGOAL = TRUE $
      N = DEP $
      END$
       END
       ELSE BEGIN
         TESTGOAL = FALSE $
         NEWGOAL(G1,G2,N) $
        END$
      END$


       PROCEDURE INSERTGOAL(GI,G2,N) $
       INTEGER GI,G2,N $

    COMMENT IF THE CURRENT STRUCTURE AT A NODE IS OLD BUT THE GOAL IS NEW
      FILE THE PAIR AT THE POSITION INDEXED BY THE INITIAL STRUCTURE $

      BEGIN
       INTEGER I,CI,VAL,NEXT,DEP $
      IF BINSEARCH(GI,I) THEN
      BEGIN
       CI = GOALIST(I,2) $
    SI:
      UNPACK3(GOALS(CI),VAL,NEXT,DEP) $
       IF NEXT NEQ 0 THEN
        BEGIN
       CI = NEXT $
        GO TO SI $
       END$
        NEXTGOAL = NEXTGOAL + 1 $
        IF NEXTGOAL GTR MAXGOALS THEN ERRORWRITE(4) $
        PACK3(VAL,NEXTGOAL,DEP,GOALS(CI)) $
        PACK3(G2,0,N,GOALS(NEXTGOAL)) $
       END
```

```
      ELSE NEWGOAL(G1,G2,N) $
      END$


      PROCEDURE NEWGOAL(G1,G2,N) $
      INTEGER G1,G2,N $

   COMMENT IF A NEW INITIAL AND GOAL STRUCTURE OCCUR AT A NODE FILE THEM

      BEGIN
      TOPGL = TOPGL + 1 $
      IF TOPGL GEQ MAXGL THEN ERRORWRITE(4) $
      GOALIST(TOPGL,1) = G1 $
      NEXTGOAL = NEXTGOAL + 1 $
      IF NEXTGOAL GEQ MAXGOALS THEN ERRORWRITE(4) $
      GOALIST(TOPGL,2) = NEXTGOAL $
      PACK3(G2,J,N,GOALS(NEXTGOAL)) $
     END$



      PROCEDURE BUILDGOAL(A,P1,J,B,P2,C,CL) $
      INTEGER ARRAY A,B,C $
      INTEGER P1,P2,J,CL $

   COMMENT PROCEDURE ESTABLISHES A SUBGOAL FOR  A STRUCTURE A  WHICH IS
    TO BE TRANSFORMED SO THAT OPERATOR B CAN BE APPLIED $

     BEGIN
      INTEGER C1,C2,C3,MAXVAR,BACK $
       IF A(J,3) NEQ 0 THEN
       BEGIN
       MAXVAR = 0 $
       FOR C1 = P2 STEP 1 UNTIL CL DO
        IF C(C1,1) LSS MAXVAR THEN
        MAXVAR = C(C1,1) $
       BACK = A(J,3) $
       P(1,1) = A(BACK,5) $
       C2 = 1 $
       FOR C1 = 1 STEP 1 UNTIL A(BACK,5) DO
        BEGIN
        IF C1 EQL A(J,2) THEN
         FOR C3 = P2 STEP 1 UNTIL CL DO
         BEGIN
          C2 = C2 + 1 $
          B(C2,1) = C(C3,1) $
       END
       ELSE BEGIN
        C2 = C2 + 1 $
        MAXVAR = MAXVAR - 1 $
        B(C2,1) = MAXVAR $
       END$
      END$
      J = BACK $
       FOR C1 = 1 STEP 1 UNTIL C2 DO
        C(C1,1) = B(C1,1) $
       P2 = 1 $
```

```
        CL = C2 $

    COMMENT IF NOT YET AT BASE OF STRUCTURE THEN BUILD THE GOAL TO ANOTHER
     LEVEL $

        BUILDGOAL(A,P1,J,B,P2,C,CL) $
        END$
       END$


       PROCEDURE COPY(A,P1,B,P2) $
       INTEGER ARRAY A,B $
       INTEGER P1,P2 $

    COMMENT COPIES ONE STRUCTURE TO ANOTHER $

      BEGIN
       INTEGER C1,C2,C3 $
       C2 = P2 $
        FOR C1 = P1 STEP 1 UNTIL A(P1,4) DO
        BEGIN
        FOR C3 = 1 STEP 1 UNTIL 5 DO
          B(C2,C3) = A(C1,C3) $
         C2 = C2 + 1 $
         END$
       END$



       PROCEDURE LINK(N1,N2) $
       INTEGER N1,N2 $

    COMMENT PROCEDURE LINKS SON N1 TO FATHER N2 BY A FIRST SON-BROTHERS
      LINK $

      BEGIN
      INTEGER C1,LAST $
       NODE(N2,1) = N1 $
       IF NODE(N1,8) EQL 0 THEN
        NODE(N1,8) = N2
         ELSE BEGIN
          LAST = NODE(N1,8) $
       FOR C1 = NODE(LAST,9) WHILE C1 NEQ 0 DO
        LAST = C1 $
       NODE(LAST,9) = N2 $
       END$
       NODE(N2,9)  = 0 $
       END$


       PROCEDURE BACKUP(NODE1,NODE2) $
       INTEGER NODE1,NODE2 $

    COMMENT PROCEDURE TAKES A NEW NODE AND ESTABLISHES ITS POSITION AS A
     POSSIBLE BEST SUCCESSOR TO ITS ANCESTORS $

      BEGIN
```

```
        INTEGER N1,BEST $
         N1 = NODE1 $
         BEST = NODE(N1,3) $
        FOR N1 = NODE(N1,1) WHILE N1 NEQ 0 DO
        BEGIN
         IF NODEVAL(BEST) LSS NODEVAL(NODE(N1,3)) THEN
          BEGIN
          BEST = NODE(1,3) $
          GO TO ST1 $
          END
         ELSE NODE(N1,3) = BEST $
     ENDS
     ST1:

     COMMENT RETURN THE BEST NODE IN TREE ELSE RETURN ZERO IF NONE EXISTS $

        IF NODEVAL(BEST) LSS ERR THEN NODE2 = 0
          ELSE NODE2 = BEST $
       ENDS



        PROCEDURE BACKONE(NODE1,BNODE) $
        INTEGER NODE1,BNODE $

     COMMENT PROCEDURE TAKES A RE-EVALUATED NODE AND ESTABLISHES ITS BEST
      SUCCESSOR AND NEW RELATION TO ITS ANCESTOR NODES $

       BEGIN
        INTEGER N1,N2,BEST $
        N1 = NODE1 $
        BEST = NODE(N1,3) $
        FOR N1 = NODE(N1,1) WHILE N1 NEQ 0 DO
        BEGIN
         N2 = NODE(N1,8) $
     ST1:
        IF NODEVAL(NODE(N2,3)) GTR NODEVAL(BEST) THEN
         BEST = NODE(N2,3) $
        N2 = NODE(N2,9) $
        IF N2 NEQ 0 THEN GO TO ST1 $

     COMMENT ESTABLISH BEST NODE IN THE TREE $

        IF NODEVAL(N1) GTR NODEVAL(BEST) THEN
        NODE(N1,3) = BEST = N1
        ELSE NODE(N1,3) = BEST $
        ENDS
        IF NODEVAL(BEST) LSS ERR THEN BNODE = 0
         ELSE BNODE = BEST $
        ENDS


        PROCEDURE INSERTOPS(N) $
        INTEGER N $

     COMMENT PROCEDURE LINKS SET OF OPERATORS TO NODE $
```

```
     BEGIN
      INTEGER C1 $
     PACK2(OPNUM,FREEOPS,NODE(N,9)) $
      FOR C1 = 1 STEP 1 UNTIL OPNUM DO
       BEGIN
       OPER(FREEOPS,1) = OPLIST(C1,1) $
       OPER(FREEOPS,2) = OPLIST(C1,2) $
       OPER(FREEOPS,3) = OPLIST(C1,3) $
       OPERVAL(FREEOPS) = OPVALUE(C1) $
       FREEOPS = OPER(FREEOPS,4) $
       IF FREEOPS EQL LASTOP THEN ERRORWRITE(6) $
      END$
      END$


      PROCEDURE FILENODE(N1,N2,LEV,STR,ENT,DEP) $
      INTEGER N1,N2,LEV,STR,ENT,DEP $

 COMMENT PROCEDURE LOADS PARAMETERS TO NODE $

     BEGIN
      NODE(NEXTNODE,2) = N1*FACTOR + N2 $
      NODE(NEXTNODE,3) = 0 $
      NODE(NEXTNODE,5) = LEV $
      NODE(NEXTNODE,10)= DEP $
      NODE(NEXTNODE,8) = 0 $
      NODEVAL(NEXTNODE) = 0.0 $
      NEXTNODE = NEXTNODE + 1 $
      IF NEXTNODE GTR MAXNODES THEN
       ERRORWRITE(9) $
     END$



      PROCEDURE EVALUATE(ANODE) $
      INTEGER ANODE $

 COMMENT PROCEDURE SSIGNS A VALUE TO A NODE ON THE BASIS OF THE DEPTH,
  SUBGOAL LEVEL AND THE VALUE OF THE OPERATORS $

      BEGIN
      INTEGER C1,C2,PNT,OPS $
      REAL R2, TOTAL $
      UNPACK2(NODE(ANODE,9),OPS,PNT) $

     COMMENT IF NO OPS SET VALUE TO ZERO $

      IF OPS EQL 0 THEN
       NODEVAL(ANODE) = 0.0
      ELSE BEGIN
       TOTAL = 0.0 $

 COMMENT SELECT <= N BEST OPERATORS AND CALCULATE AVERAGE VALUE $

      IF OPS GTR EVALOPS THEN C2 = EVALOPS
       ELSE C2 = OPS $
       FOR C1 = 1 STEP 1 UNTIL C2 DO
```

```
          BEGIN
          TOTAL = TOTAL + OPERVAL(PNT) $
          PNT = OPER(PNT,4) $
          END$
          TOTAL = TOTAL/C2 $
     S1:

     COMMENT ADD FACTORS FOR DEPTH AND SUBGOAL LEVEL $

         R2 = NODE(ANODE,10) + 1 $
         TOTAL = TOTAL - R2/EVALDEPTH $
         R2 = NODE(ANODE,5) + 1 $
         TOTAL = TOTAL + R2/EVALSUB $
         NODLVAL(ANODE) = TOTAL $
         END$
        C1 =  ANODE $

     COMMENT RESET THE BEST SUCCESSOR TO THE NODE $

        PNT = NODE(ANODE,8) $
        IF PNT EQL 0 THEN
        NODE(ANODE,3) = ANODE
        ELSE
         BEGIN
     ST1:
        IF NODEVAL(NODE(PNT,3)) GTR NODLVAL(C1) THEN
         C1 = NODE(PNT,3) $
         PNT = NODE(PNT,9) $
         IF PNT NEQ 0 THEN GO TO ST1 $
         NODE(ANODE,3) = C1 $
        END$
        END$


        PROCEDURE ERRORWRITE(K) $
        INTEGER K $

     COMMENT PROCEDURE OUTPUTS ERROR MESSAGE AND EITHER SELECTS NEXT PROBLE
      OR HALTS EXECUTION $

        BEGIN
         SWITCH ERROR = E1,E2,E3,E4,E5,E6,E7,E8,E9,E10 $
         GO TO ERROR(K) $
     E1:
       WRITE('MAXIMUM STORAGE IN CANONICAL TREE EXCEEDED') $
        GO TO MAINEND $
     E2:
        WRITE('INCORRECT ADDRESS FOR STRING RETRIEVAL') $
        GO TO MAINEND $
     E3:
        WRITE('MAXIMUM NUMBER OF GOALS EXCEEDED') $
        GO TO MAINEND $
     E4:
        WRITE('MAXIMUM NUMBER OF HELD SUBGOALS EXCEEDED') $
        GO TO MAINEND $
     E5:
        WRITE('ERROR IN INPUT DATA') $
```

```
        GO TO E11 $
    E6:
        WRITE('MAXIMUM NUMBER OF OPERATORS GENERATED') $
        GO TO MAINEND $
    E7:
        WRITE('MAXIMUM STRUCTURE SIZE EXCEEDED') $
        GO TO E11 $
    E8:
        WRITE('TOO MANY SYMBOLS DEFINED') $
        GO TO MAINEND $
    E9:
        WRITE('MAXIMUM NUMBER OF NODES GENERATED') $
        GO TO MAINEND $
    E10:
        WRITE('TOO MANY OPERATORS DEFINED') $
        GO TO MAINEND $
    E11:
       END$



        PROCEDURE POSMAP(A,P1,L1) $
         INTEGER ARRAY A $
         INTEGER P1,L1 $

    COMMENT PROCEDURE TRANSFORMS POLISH STRING INTO TREE STRUCTURE BY
     SETTING UP BACKWARD AND FORWARD LINKS $

       BEGIN
         INTEGER ARRAY STACK(1:MAXSTAX,1:5) $
         INTEGER PT1,PT2,DEG,C1 $
         A(P1,5) = A(P1,2) = A(P1,3) = 0 $
         A(P1,4) = P1 $
         IF OPERATOR(A(P1,1),DEG) THEN
         BEGIN

    COMMENT IF ELEMENT IS AN OPERATOR TACK IT WITH DEGREE AND CURRENT
     POSITION $

          A(P1,5) = DEG $
          PT2 = 1 $
          STACK(PT2,1) = 0 $
          STACK(PT2,2) = STACK(PT2,3) = DEG $
          STACK(PT2,4) = P1 $
          STACK(PT2,5) = 1 $
         FOR PT1 = P1+1 STEP 1 UNTIL L1 DO
         BEGIN

    COMMENT DECREMENT TOP OF STACK DEGREE AND INSERT BACK POINTER $

          STACK(PT2,2) = STACK(PT2,2) - 1 $
          A(PT1,2) =  (STACK(PT2,3) - STACK(PT2,2)) $
          A(PT1,3) = STACK(PT2,4) $
          IF OPERATOR(A(PT1,1),DEG) THEN
           BEGIN

    COMMENT IF OPERATOR STACK IT WITH DEGREE $
```

```
            PT2 = PT2 + 1 $
            STACK(PT2,2) = STACK(PT2,3) = DEG $
            STACK(PT2,1) = A(PT1,2) $
            STACK(PT2,4) = PT1 $
            STACK(PT2,5) = 0 $
            A(PT1,5) = DEG $
           END

      COMMENT IF OPERAND INSERT FORWARD POINTER TO SELF

           ELSE BEGIN A(PT1,4) = PT1 $
         A(PT1,5) = 0 $
         END$
           FOR C1 = 1 STEP 1 UNTIL PT2 DO
            STACK(C1,5) = STACK(C1,5) + 1 $
     S1:

     COMMENT IF ALL OPERANDS OF TOP OPERATOR HAVE BEEN DEALT WITH SELECT
      NEXT LOWER OPERATOR $

           IF STACK(PT2,2) LEQ 0 THEN
            BEGIN
            A(STACK(PT2,4),4) = STACK(PT2,4) + STACK(PT2,5) - 1 $
            PT2 = PT2 - 1 $
            IF PT2 NEQ 0 THEN GO TO S1 $
            END$
           END$
          END$
         END$


          BOOLEAN PROCEDURE TERMSPEC(N1,N2) $
          INTEGER N1,N2 $

     COMMENT PROCEDURE DETERMINES WHETHER CONSTANTS N1 AND N2 ARE EQUIVALENT
      BY CHECKING THAT THEY BELONG TO THE SAME CONSTANT CLASS OR IF EITHER
       IS A VARIABLE OPERATOR,THAT THEY HAVE THE SAME DEGREE $

         BEGIN
          IF N1 NEQ N2 THEN
           BEGIN
           IF SYMTAB(N1,1) NEQ SYMTAB(N2,1) THEN
            TERMSPEC = FALSE
            ELSE BEGIN
           IF SYMTAB(N1,2) EQL SYMTAB(N2,2) THEN
            TERMSPEC = TRUE
             ELSE BEGIN
             IF SYMTAB(N1,2) EQL 0 OR SYMTAB(N2,2) EQL 0 THEN
              TERMSPEC = TRUE
             ELSE TERMSPEC = FALSE $
             END
            END
           END
           ELSE TERMSPEC = TRUE $
          END$
```

```


     BOOLEAN PROCEDURE CORRELT(A,P1,L1,B,P2,L2) $
     INTEGER ARRAY A,B $
     INTEGER P1,L1,P2,L2 $

COMMENT PROCEDURE DETERMINES IF ELEMENT P1 OF A CORRESPONDS TO ELEMENT
  P2 OF B $

    BEGIN
     INTEGER N1,N2 $
     IF ROUGHMATCH(A,P1,L1,B,P2,L2) THEN
      BEGIN

COMMENT IF THE ELEMENTS ROUGHLY MATCH THEN DETERMINE IF EACH LINK IS A
   SPECIFICATION OF ITS COUNTERPART $

      IF P1 NEQ L1 THEN
        BEGIN
      N1 = L1 $
      N2 = L2 $
S1:
     N1 = A(N1,3) $
     N2 = B(N2,3) $
     IF NOT TERMSPEC(A(N1,1),B(N2,1)) THEN GO TO FAIL $
     IF P1 NEQ N1 THEN GO TO S1 $
  END$
      CORRELT = TRUE $
     END
    ELSE
FAIL:
      CORRELT = FALSE $
    END$



     BOOLEAN PROCEDURE ROUGHMATCH(A,P1,L1,B,P2,L2) $
     INTEGER ARRAY A,B $
     INTEGER P1,L1,P2,L2 $

COMMENT PROCEDURE DETERMINES WHETHER ELEMENT P1 OF A ROUGHLY MATCHES
  TO ELEMENT P2 OF B $

    BEGIN
     INTEGER ARRAY CSTACK(1:MAXOPS,1:2) $
     INTEGER C1,C2,C3,PT $
     ROUGHMATCH = TRUE $
     L2 = P2 $
     IF P1 EQL L1 THEN GO TO S2 $
     PT = L1 $
     C1 = 0 $
LOOP1:

COMMENT STACK BACKWARD LINKS OF A TOGETHER WITH DEGREE OF EACH OP $

     C1 = C1 + 1 $
     CSTACK(C1,1) = A(PT,2) $
```

```
      CSTACK(C1,2) = A(PT,5) $
      IF P1 NEQ PT THEN BEGIN
       PT = A(PT,3) $
       GO TO LOOP1 $
      END $
      PT = P2 $

   COMMENT MATCH FORWARD IN B USING LINKS IN STACK. IF MATCH IS NOT
    POSSIBLE THEN FAIL $

      FOR C2 = C1 STEP -1 UNTIL 2 DO
       BEGIN
       IF B(PT,5) NEQ CSTACK(C2,2) THEN GO TO RFAIL $
        PT = PT + 1 $
       FOR C3 = 2 STEP 1 UNTIL CSTACK(C2-1,1) DO
        PT = B(PT,4) + 1 $
       END $
      L2 = PT  $
      GO TO S2 $
   RFAIL:
      ROUGHMATCH = FALSE $
   S2:
     END $



      BOOLEAN PROCEDURE OPERATOR(P,L) $
      INTEGER P,L $

   COMMENT PROCEDURE DETERMINES WHETHER A CONSTANT IS AN OPERATOR AND
    RETURNS ITS DEGREE $

     BEGIN
      OPERATOR = FALSE $
      IF P GTR 0 THEN
       BEGIN
       IF SYMTAB(P,1) GTR 0 THEN
        BEGIN
        OPERATOR = TRUE $
        L = SYMTAB(P,1) $
        END $
       END $
      END $


      PROCEDURE DIFFCHECK(G,K,P) $
      INTEGER G,K,P $

   COMMENT PROCEDURE INSERTS NEW DIFFERENCES INTO THE DIFFERENCE SET $

     BEGIN
      INTEGER C1 $
      FOR C1 = 1 STEP 1 UNTIL DIFFNUM DO
       IF G EQL DIFFS(C1,1) AND K EQL DIFFS(C1,2) THEN
        GO TO DEND $
      DIFFNUM = DIFFNUM + 1 $
      DIFFSET(P) = DIFFSET(P) + 1 $
```

```
          DIFFS(DIFFNUM,1) = J $
          DIFFS(DIFFNUM,2) = K $
       END;
         END$



          PROCEDURE ZERODIFFS(A,B,S,G) $
          INTEGER ARRAY A,B $
          INTEGER S,G $

       COMMENT PROCEDURE DETERMINES THE SET OF ZERO-LEVEL DIFFERENCES BETWEEN
         STRUCTURE A AND B ,ROOTED AT S AND G RESPECTIVELY $

         BEGIN
          INTEGER ARRAY STACK(1:MAXSTAX),SVAR(1:MAXSTAX,1:2) $
          INTEGER P1,P2,CN,V1,V2,C1,C2,CT,P $
          CN = P = 0 $
          P2 =V1 =V2 =0 $
          FOR P1 = S STEP 1 UNTIL A(S,4) DO
           BEGIN

       COMMENT IF THERE IS A CORRESPONDING CONSTANT ELEMENT BETWEEN A&B THEN
        TEST IF THEY ARE EQUIVALENT $

            IF CORRELT(A,S,P1,B,G,P2) THEN
            BEGIN
             IF B(P2,1) GTR 0 THEN
             BEGIN
              IF NOT DIFFSPEC(A(P1,1),B(P2,1),SVAR,CN) THEN
               DIFFCHECK(B(P2,1),P1,P) $
             END
             ELSE IF A(P1,1) GEQ 0 THEN
             BEGIN

       COMMENT IF THE CORRESPONDING ELEMENT IN B IS VARIABLE,SUBSTITUTE THE
       MATCH IN A. IF THIS VARIABLE OCCURS IN MORE THAN ONE POSITION TEST
        THE RESULTING STRUCTURE AGAINST A FOR DIFFERENCES $

              CT = 0 $
              FOR C1 = G STEP 1 UNTIL B(G,4) DO
              IF B(C1,1) EQL B(P2,1) AND C1 NEQ P2 THEN
              BEGIN
               CT = CT + 1 $
               STACK(CT) = C1 $
              END$
              IF CT GTR 0 THEN
               FOR C1 = 1 STEP 1 UNTIL CT DO
               BEGIN
                IF CORRELT(B,G,STACK(C1),A,S,V1) THEN
                FOR C2 = P1 STEP 1 UNTIL A(P1,4) DO
                 BEGIN
                 IF CORRELT(A,P1,C2,A,V1,V2) THEN
                  BEGIN
                  IF NOT DIFFSPEC(A(C2,1),A(V2,1),SVAR,CN) THEN
                   DIFFCHECK(A(C2,1),V2,P) $
                  END$
```

```
                END$
               END$
              END$
            END$
             END$
           END$



      BOOLEAN PROCEDURE DIFFSPEC(A,B,VAR,CN) $
      INTEGER ARRAY VAR $
      INTEGER A,B,CN $

  COMMENT PROCEDURE DETERMINES WHETHER TWO ELEMENTS ARE EQUIVALENT $

     BEGIN
      INTEGER DEG,C1 $

  COMMENT IF B VARIABLE THEN AUTOMATIC MATCH $
      IF B LSS 0 THEN
       DIFFSPEC = TRUE
        ELSE BEGIN

  COMMENT IF A VARIABLE THEN ONLY CERTAIN SUBSTITUTIONS ARE VALID $

       IF A LSS 0 THEN BEGIN
        IF OPERATOR(B,DEG) THEN DIFFSPEC = FALSE
          ELSE BEGIN

  COMMENT DETERMINE FOR ZERODIFFS IF THIS SUBSTITUTION IS VALID $

      FOR C1 = 1 STEP 1 UNTIL CN DO
       IF VAR(C1,1) EQL A THEN
        BEGIN
         IF VAR(C2,2) EQL B THEN
          DIFFSPEC = TRUE
          ELSE DIFFSPEC = FALSE $
          GO TO S1 $
        END$
       CN = CN + 1 $
       VAR(CN,1) = A $
        VAR(CN,2) = B $
        DIFFSPEC = TRUE $
       GO TO S1 $
       END
     IF BOTH A & B ARE CONSTANTS DO CONSTANT TEST $
       END
        ELSE IF TERMSPEC(A,B) THEN DIFFSPEC = TRUE
         ELSE DIFFSPEC = FALSE $
     END$
  S1:
     END$



      PROCEDURE ORDEROPS $

```

```
      COMMENT PROCEDURE ORDERS THE OPERATORS IN DECREASING VALUE $

          BEGIN
          INTEGER C1,C2,C3,TEMP1,TEMP2,TEMP3 $
          REAL  TEMPVAL $
          FOR C1 = 1 STEP 1 UNTIL (OPNUM-1) DO
           BEGIN
            C2 = C1 $
            FOR C3 = C1+1 STEP 1 UNTIL OPNUM DO
             IF OPVALUE(C3) GTR OPVALUE(C2) THEN
              C2 = C3 $
             IF C2 NEQ C1 THEN
             BEGIN
              TEMP1 = OPLIST(C1,1) $
              TEMP2 = OPLIST(C1,2) $
              TEMP3 = OPLIST(C1,3) $
              TEMPVAL = OPVALUE(C1) $
              OPLIST(C1,1) = OPLIST(C2,1) $
              OPLIST(C1,2) = OPLIST(C2,2) $
              OPLIST(C1,3) = OPLIST(C2,3) $
              OPVALUE(C1) = OPVALUE(C2) $
              OPLIST(C2,1) = TEMP1 $
              OPLIST(C2,2) = TEMP2 $
              OPLIST(C2,3) = TEMP3 $
              OPVALUE(C2) = TEMPVAL $
             END$
            END$
          C1 = 0 $
          FOR C1=C1+1 WHILE C1 LEQ OPNUM DO
           IF OPVALUE(C1) LSS ERR THEN OPNUM = C1-1 $
           END$

          PROCEDURE OPCHECK(I,J,P,D,A,P1,ELT) $
          INTEGER I,J,P,D,ELT,P1 $
          INTEGER ARRAY A $

      COMMENT PROCEDURE EVALUATES AND FILLS A NEW OPERATOR. IF IT HAS BEEN
       GENERATED PREVIOUSLY IT UPDATES THE VALUE $

         BEGIN
          INTEGER C1,C2,TEMP,PT $
          REAL DIFF1,DIFF2,DP,D1 $
          DIFF1 = (RULESL(I,2)-RULESL(I,1)) - (RULESR(I,2)-RULESR(I,1)) $
          DIFF2 = D $
          DP = P $
          TEMP = I*FACTOR + J $
          PT = P1 + J $

      COMMENT DETERMINE WHETHER NEW OR OLD OPERATOR $

          FOR C1 = 1 STEP 1 UNTIL OPNUM DO
           IF OPLIST(C1,1) EQL TEMP THEN GO TO F1 $

      COMMENT IF NEW INSERT TO LIST AND GIVE VALUE BASED ON LEVEL AND
       COMPLEXITY $

          OPNUM = OPNUM + 1 $
```

```
      OPLIST(OPNUM,1) = TEMP $
      OPLIST(OPNUM,2) = ELT $
      OPVALUE(OPNUM) = DEPTHBIAS1/(DEPTHBIAS2*DP + DEPTHBIAS4) + RCOMP(I)*
      COMPBIAS $

COMMENT IF STRUCTURE IS SPEC OF OP INPUT ADD SPECBIAS ELSE ADD A
 VALUE BASED ON AMOUNT OF WORK REQUIRED $

      IF SPECIFICATION(A,RULE,PT,RULESL(I,1)) THEN
    BEGIN
    OPLIST(OPNUM,3) = 1 $
      OPVALUE(OPNUM) = OPVALUE(OPNUM) + SPECBIAS $
  END
     ELSE BEGIN
    OPLIST(OPNUM,3) = 0 $
     D1 = DIFFEVAL(RULE,RULESL(I,1),A,PT) $
     OPVALUE(OPNUM) = OPVALUE(OPNUM) + D1 $
     END$

COMMENT ADD FACTORS FOR DIFFERENCE IN SIZE PLUS WHETHER OP TRANSFORMS
 STRUCTURE TOWARDS REQUIRED SIZE $

     OPVALUE(OPNUM) = OPVALUE(OPNUM) +DIFFBIAS1/(ABS(DIFF1-DIFF2) +
      DIFFBIAS2) $
     DIFF1 = RULESL(I,2) -RULESL(I,1)+1 $
     DIFF2 = A(PT,4) - PT + 1 $
OPVALUE(OPNUM)=OPVALUE(OPNUM)+DIFFBIAS3/(ABS(DIFF1-DIFF2) + DIFFBIAS4)
     GO TO F2 $
F1:

COMMENT ADD FACTOR TO INCREASE VALUE OF OLD OP WHICH REMOVES ANOTHER
   DIFFERENCE $

     OPVALUE(C1) = OPVALUE(C1) + DEPTHBIAS1/(DEPTHBIAS3*DP + DEPTHBIAS4)
F2:
   END$



     REAL PROCEDURE DIFFEVAL(A,P1,B,P2) $
     INTEGER ARRAY A,B $
     INTEGER P1,P2 $

COMMENT PROCEDURE DERIVES A FACTOR TO REFLECT THE PROBABLE AMOUNT OF
 WORK REQUIRED TO MAKE AN OPERATOR APPLICABLE $

   BEGIN
     REAL D $
     INTEGER C1,C2,CN $
    INTEGER ARRAY VAR(1:MAXSTAX,1:2) $
     D = 0.3 $

COMMENT LOAD FACTOR IF THE BASES DIFFER $

     IF NOT DIFFSPEC(A(P1,1),B(P2,1),VAR,CN) THEN
      D = DIFFACTOR1 $
     C1 = P1 $
```

```
     C2 = P2 $
   LOOP:

   COMMENT COUNT POSITIONS OF DIFFERENCE $

      IF NOT DIFFSPEC(A(C1,1),B(C2,1),VAR,CN) THEN
      D = D + 1.0 $
     IF A(C1,5) NEQ B(C2,5) THEN
       BEGIN
       C1 = A(C1,4) $
       C2 = B(C2,4) $
       END$
      C1 = C1 + 1 $
      C2 = C2 + 1 $
      IF C1 LEQ A(P1,4) AND C2 LEQ B(P2,4) THEN GO TO LOOP $
     DIFFEVAL = DIFFACTOR2/D $
    END$



     PROCEDURE OPDIFFGENERATE(A,B,P1,P2,P,LEV,NN) $
     INTEGER ARRAY A,B $
     INTEGER P1,P2,P,NN,LEV $

   COMMENT PROCEDURE ACCEPTS A SET OF ZERO-LEVEL DIFFERENCES AND GENERATES
    SETS OF HIGHER LEVEL DIFFERENCES AND OPERATORS $

     BEGIN
      INTEGER ARRAY CSTACK(1:MAXSTAX,1:2),OPTEST(1:MAXSUBGOALS,1:4),
     SVAR(1:MAXSTAX,1:2),OPPOS(1:100),STACK1(1:6) $
      INTEGER PJ,POINT,PT,PG,PK,D,DIFFLEVEL,NEXTDIFF,PTR,PR,PS,C1,C2,PNT,
    C3,C4,C6,MAP,PL,CH,ELT,C5,TTP,PX,OP,SUBOPS,POS,BACK,I,J,OPLEVEL,L,K,N
     BOOLEAN FLAG1,FLAG2 $
     FLAG1 = FALSE $
      IF LEV LSS MAXSUBGOALS THEN
       BEGIN

   COMMENT SECTION ASSISTS IN RESTRICTING OPERATOR GENERATION BY KEEPING
    TRACK OF THE PURPOSE OF SUBGOALS, POSITION OF APPLICATION AND THE
    'ESSENTIAL' ELEMENT (S) OF EACH OPERATOR GENERATED THE SUBGOALS ON
    THIS PATH ARE NOTED $

       SUBOPS = 0 $
       POS = 1 $
       BACK = NN $
       OPLEVEL = LEV $
      FOR SUBOPS = SUBOPS + 1 WHILE OPLEVEL NEQ MAXSUBGOALS DO
         BEGIN
         FOR PN=PN+1 WHILE NOT(NODE(BACK,5) EQL OPLEVEL AND
          NODE(NODE(BACK,1),5) GTR OPLEVEL) DO  BACK = NODE(BACK,1) $
          UNPACK3(NODE(BACK,7),1,PNT,N) $
          J = P1 $

   COMMENT IF OP WITHIN FIRST LEVEL OF STRUCTURE DETERMINE POSITION DIRECT

      IF J EQL 0 THEN
       BEGIN
```

```
          IF PNT LSS 2 THEN J = J + PNT
         ELSE BEGIN
         J = J + 1 $
         FOR C3 =(2,1,PNT)DO J = A(J,4) + 1 $
         END $
         OPTEST(SUBOPS,3) = J $

COMMENT IF OP TO BE APPLIED AT BASE SET NEGATIVE FLAG ELSE INSERT ITS
 POSITION IN THE STRUCTURE $

         IF J EQL P1 THEN OPTEST(SUBOPS,2) = -1
          ELSE BEGIN
           OPTEST(SUBOPS,2) = 1 $
           OPTEST(SUBOPS,1) = POS $
           OPPOS(POS) = PNT $
         POS = POS + 1 $
         END
          END
         ELSE BEGIN

COMMENT IF OP BELOW FIRST LEVEL THEN SET UP STACK OF LINKS TO IDENTIFY
POSITION $

          PNT = PNT - 1 $
         OPTEST(SUBOPS,1) = POS $
         OPTEST(SUBOPS,2) = N $
          FOR PNT = PNT + 1 WHILE N GTR 0 DO
           BEGIN
           IF N GTR 6 THEN C1 = 6
             ELSE C1 = N $
            UNPACK6(STACK1,SUBOPLIST(PNT)) $
            FOR C2=(1,1,C1) DO
             BEGIN
              J = J + 1 $
              OPPOS(POS) = STACK1(C2) $
              POS = POS + 1 $
              FOR C3 = (2,1,STACK1(C2)) DO
               J = A(J,4) + 1 $
             END$
            N = N-6 $
            END$

COMMENT SET POSITION OF ESSENTIAL ELEMENT $

       OPTEST(SUBOPS,3) = J $
          END$

COMMENT IF NO SUCH ELEMENT SET FLAG NEGATIVE OTHERWISE SET UP STACK OF
 POINTERS $

         IF NODE(BACK,6) LSS 0 THEN OPTEST(SUBOPS,4) = NODE(BACK,6)
          ELSE BEGIN
          UNPACK6(STACK1,NODE(BACK,6)) $
          OPTEST(SUBOPS,4) = STACK1(6) $
          FOR C1 =(1,1,STACK1(6)) DO
           BEGIN
           OPPOS(POS) = STACK1(C1) $
```

```
          POS = POS + 1 $
          END$
         END$
         OPLEVEL = OPLEVEL + 1 $
         BACK = NODE(BACK,1) $
        END$
        SUBOPS = SUBOPS - 1$
        FOR C1=(1,1,SUBOPS) DO
         IF OPTEST(C1,2) GTR 0 OR OPTEST(C1,4) GEQ 0 THEN FLAG1 = TRUE $
        END$
        D = ((A(P1,4)-P1)-(B(P2,4)-P2)) $
        OPNUM = 0 $
        PN = PR = 0 $

     COMMENT SET INITIAL VALUES FOR DIFFERENCE SETS AND FIRST DIFF $

        PT = 0 $
        DIFFLEVEL = DIFFNUM $
        NEXTDIFF = 1 $
     S1:
        IF NEXTDIFF GTR DIFFNUM THEN GO TO OPDEND $

     COMMENT SELECT NEXT DIFFERENCE AND STACK ITS POSITION IN RELATION
     TO THE BASE $

        PG = DIFFS(NEXTDIFF,1) $
        PK = DIFFS(NEXTDIFF,2) $
         PL = PR = PK $
        PTR = 0 $
     L1:
        PTR = PTR + 1 $
        CSTACK(PTR,1) = PR $
        CSTACK(PTR,2) = A(PL,2) $
        PL = PR $
        PR = A(PR,3) $
        IF PR NEQ 0 THEN GO TO L1 $

     COMMENT BEGIN MAIN LOOP FOR ALL OPERATORS $

        FOR C1 = 1 STEP 1 UNTIL RULENO DO
         BEGIN
         TTP = PL = TERMTABENTRY(C1) $
         FOR C2 = 1 STEP 1 UNTIL PTR DO
          BEGIN

     COMMENT SELECT POINT OF APPLICATION OF OPERATOR $

          PR = CSTACK(C2,1) $
          FOR C3 = 1 STEP 1 UNTIL C2 DO
           BEGIN

     COMMENT SELECT POINT OF DIFFERENCE OR STRUCTURE CONTAINING POINT OF
     DIFFERENCE $

          PS = CSTACK(C3,1) $
          PN = CSTACK(C2,1) $
          FLAG2 = FALSE $
```

```
        TTP = PL $

COMMENT WORK FORWARD IN OPTABLE TO POINT OF DIFFERENCE. IF AT ANY
 STAGE THE FORWARD STEP IS NOT POSSIBLE I.T.O. MATCHING OR END OF OP
  THEN TRY AT NEXT POINT OF APPLICATION $

    FOR C4 =(C2,-1,C3+1) DO
     BEGIN
      PN = CSTACK(C4,1) $
      IF TERMTAB(TTP,4) LSS 0 THEN GO TO NEXT1 $
      IF NOT TERMSPEC(TERMTAB(TTP,4),A(PN,1)) THEN GO TO NEXT1 $
    TTP = TERMTAB(TTP,2) $
    FOR C5 = (2,1,CSTACK(C4,2)) DO
     IF TTP EQL 0 THEN GO TO NEXT1
       ELSE TTP = TERMTAB(TTP,3) $
    IF TTP EQL 0 THEN GO TO NEXT1 $
   ENDS

COMMENT IF THE POSITION IS UNALTERED THEN EXIT $

    IF TERMTAB(TTP,1)  EQL 0 THEN GO TO NEXT1 $
     IF TERMTAB(TTP,1) GTR 0 THEN
      BEGIN

COMMENT IF DIFF MATCHES A CONSTANT THEN TEST DIRECTLY $

      ELT = -1 $
      IF DIFFSPEC(PG,TERMTAB(TTP,1),SVAR,CN) THEN GO TO CHECK2 $
       GO TO NEXT1 $
      END
     ELSE BEGIN

COMMENT IF DIFF MATCHES A VARIABLE THEN ISOLATE MATCH IN CURRENT
  STRUCTURE $

      PQ = -TERMTAB(TTP,1) $
      FLAG2 = TRUE $
L2:
     PN = CSTACK(C2,1)  $
     C4 = C2+1$
     FOR C4 = C4-1 WHILE PN GTR 0 DO
     BEGIN
       IF A(PN,1) NEQ VARTAB(PQ,1) THEN GO TO CHECK1 $
      PN = PN + 1$
      FOR C5 = (2,1,VARTAB(PQ,3)) DO PN = 1 + A(PN,4) $
     PQ = VARTAB(PQ,4) $
     ENDS

COMMENT
IF VARIABLE MATCHES SUBSTRUCTURE CONTAINING DIFFPOINT THEN MATCH
   WITHIN SUBSTRUCTURE TO DETERMINE CORRECT CORRESPONDING ELEMENT $

     IF PS NEQ PK THEN
      BEGIN
      IF NOT CORRELT(A,PS,PK,A,PN,POINT) THEN GO TO CHECK1 $
      PN = POINT $
      ENDS
```

```
          ELT = PN - P1 $

      COMMENT IF THE ELEMENT IS NOT A SPEC OF THE GOAL THEN INSET A HIGHER
       LEVEL DIFFERENCE $

          IF NOT DIFFSPEC(PG,A(PN,1),SVAR,CN) THEN
           DIFFCHECK(PG,PN,PT+1)
         ELSE BEGIN
      CHECK2:
         IF FLAG1 THEN
          BEGIN

      COMMENT IF NECESSARY DETERMINE WHETHER OPERATOR NEGATES SUBGOALS BY
       CHECKING AGAINST EACH STACKED OPERATOR AND ELEMENT $

          J = PR $
          FOR C4=(1,1,SUBOPS) DO
           BEGIN
           POS = OPTEST(C4,1) $
           IF (OPTEST(C4,3) GEQ J AND OPTLST(C4,3) LEQ A(J,4)) THEN
            BEGIN
            IF OPTEST(C4,2) GTR 0 AND OPTEST(C4,3) GTR J  THEN
             BEGIN
             TTP = TERMTABENTRY(C1) $
             I = OPTEST(C4,3) $
             L = K = OPTEST(C4,1) + OPTEST(C4,2) - 1$
             FOR I= A(I,3) WHILE I NEQ PR DO L = L-1 $
             FOR C5 = (L,1,K) DO
              BEGIN
              IF TERMTAB(TTP,4) LSS 0 THEN
                BEGIN
                IF TERMTAB(TTP,1) LSS 0 THEN GO TO CHECK1
                 ELSE GO TO CHECK3 $
                END$
               TTP = TERMTAB(TTP,2) $
             FOR C6=(2,1,OPPOS(C5)) DO
              IF TTP EQL 0 THEN GO TO CHECK1
              ELSE TTP = TERMTAB(TTP,3) $
             IF TTP EQL 0 THEN GO TO CHECK1 $
             END$
         IF TERMTAB(TTP,1) LSS 0 THEN GO TO CHECK1 $
         END$
         L = K + OPTEST(C4,4) - 1$
         IF OPTEST(C4,4) GTR 0 THEN
          BEGIN
          FOR C5=(K+1,1,L) DO
            BEGIN
            IF TERMTAB(TTP,4) LSS 0 THEN
             BEGIN
             IF TERMTAB(TTP,1) LSS 0 THEN GO TO CHECK1
              ELSE GO TO CHECK3 $
             END$
            TTP = TERMTAB(TTP,2) $
            FOR C6=(2,1,OPPOS(C5)) DO
             IF TTP EQL 0 THEN GO TO CHECK1
              ELSE TTP = TERMTAB(TTP,3) $
             IF TTP EQL 0 THEN GO TO CHECK1 $
```

```
          END$
         IF TERMTAB(TTP,1) NEQ 0 THEN GO TO CHECK1 $
        END$
        END$
   CHECK3:
       END$
       END$

   COMMENT INSERT THE OPERATOR TO ITS CORRECT SET $

       OPCHECK(C1,PR-P1,PT,D,A,B1,ELT) $
    END$
   CHECK1:
     IF FLAG2 THEN BEGIN
       IF VARTAB(PQ,4) LSS 0 THEN BEGIN
         PQ = PQ + 1 $
         GO TO L2 $     END$
     END$
     END$
   NEXT1:
       END$
       END$
   NEXTRULE1:
       END$
   COMMENT
     INCREMENT LEVEL OF OPERATOR/DIFFERENCE IF NECESSARY AND TEST IF
      MAXLEVEL EXCEEDED. $

        IF NEXTDIFF EQL DIFFLEVEL THEN
         BEGIN
         PT = PT + 1 $
         IF PT GTR P THEN GO TO OPDEND $
         DIFFLEVEL = DIFF NNN $
     END$
     NEXTDIFF = NEXTDIFF + 1 $
         GO TO S1 $
   OPDEND:
     END$



      BOOLEAN PROCEDURE SPECIFICATION(A,B,P1,P2) $
      INTEGER ARRAY A,B $
      INTEGER P1,P2 $

   COMMENT PROCEDURE TESTS WHETHER STRUCTURE A IS A SUBSTITUTION
    INSTANCE OF B $

     BEGIN
      INTEGER ARRAY STACK(1:MAXSTAX,1:2),SVAR(1:MAXSTAX,1:2) $
      INTEGER C1,C2,C3,C4,C5,CT1,CT2 $

   COMMENT IF BOTH STRUCTURES HAVE SIZE ONE DO BRIEF TEST $

      IF A(P1,4) EQL P1 AND B(P2,4) EQL P2 THEN
       BEGIN
       IF A(P1,1) LSS 0 OR B(P2,1) LSS 0 THEN
```

```
            SPECIFICATION = TRUE
           ELSE SPECIFICATION = TERMSPEC(A(P1,1),B(P2,1)) $
           GO TO QUIK $
           END$
         SPECIFICATION = TRUE $
         C1 = P1 $
         C2 = P2 $
         CT1 = CT2 = 0 $
      L1:

      COMMENT IF OF DIFFERING DEGREE THEN EXIT WITH FAILURE $

         IF A(C1,2) NEQ B(C2,2) THEN BEGIN
         IF C1 NEQ P1 THEN GO TO FAIL $
         END$
         IF B(C2,1) GTR 0 THEN BEGIN

      COMMENT IF CONSTANT IN B TEST FOR SPEC OF INDIVIDUAL ELEMENTS,$

          IF NOT DIFFSPEC(A(C1,1),B(C2,1),SVAR,CT1) THEN
           GO TO FAIL $
          C1 = C1 + 1 $
          C2 = C2 + 1 $
          END
         ELSE BEGIN

      COMMENT IF VARIABLE FIND ITS SUBSTITUTION VALUE AND TEST WHETHER
       ANOTHER IDENTICAL VARIABLE HAS BEEN SUBSTITUTED TO, IF SO TEST
      THAT SUBSTITUTION VALUES ARE EQUIVALENT $

          FOR C3 = 1 STEP 1 UNTIL CT2 DO
          IF B(C2,1) EQL STACK(C3,1) THEN
           BEGIN
           C4 = STACK(C3,2) $
           FOR C5 = C1 STEP 1 UNTIL A(C1,4) DO
            BEGIN
             IF A(C5,1) NEQ A(C4,1) THEN GO TO FAIL $
             C4 = C4 + 1 $
            END$
           GO TO L2 $
           END$
          CT2 = CT2 + 1 $
          STACK(CT2,1) = B(C2,1) $
          STACK(CT2,2) = C1 $
      L2:
         C2 = C2 + 1 $
         C1 = A(C1,4) + 1 $
         END$
         IF C1 LEQ A(P1,4) AND C2 LEQ B(P2,4) THEN GO TO L1 $
         IF C1 NEQ (A(P1,4)+1) OR C2 NEQ (B(P2,4)+1) THEN
      FAIL:
            SPECIFICATION = FALSE $
      QUIK:
         END$
```

```
      PROCEDURE POLISHINFIX(A,P) $
      INTEGER ARRAY A $
      INTEGER P $

  COMMENT PROCEDURE TRANSFORMS INTERNAL STRUCTURE TO  INFIX
   FORM FOR LEGIBILITY AND PRINTS THE INFIX FORM ,$

     BEGIN
      INTEGER ARRAY DUMMY(1:120),STACK(1:MAXSTAX,1:2) $
      INTEGER C1,C2,C3,POINT,P1,P2 $
      STRING BUFFER(360),BUFF2(120) $
      FORMAT F1(S120,A1) $
      P1 = P2 = 0 $
      FOR C1 = P STEP 1 UNTIL A(P,4) DO
      BEGIN
       IF A(C1,1) LSS 0 OR SYMTAB(A(C1,1),1) EQL 0 THEN
       BEGIN

  COMMENT INSERT OPERANDS TO DUMMY AND DECREMENT OP DEGREE IN STACK $

        P2 = P2 + 1 $
        DUMMY(P2) = A(C1,1) $
        IF P1 NEQ 0 THEN STACK(P1,2) = STACK(P1,2) - 1 $
        GO TO OPCHECK $
       END$
   COMMENT INSERT OPERATORS AND DEGREE TO STACK $

       P1 = P1 + 1 $
       STACK(P1,1) = A(C1,1) $
       STACK(P1,2) = SYMTAB(A(C1,1),1) $
  OPCHECK:
      IF P1 NEQ 0 THEN
       BEGIN

  COMMENT IF STACK DEGREE AT ZERO INSERT TO DUMMY AND DECREMENT
   STACK POINTER, $

       IF STACK(P1,2) EQL 0 THEN
        BEGIN
        P2 = P2 + 1 $
        DUMMY(P2) = STACK(P1,1) $
        P1 = P1 - 1 $
        IF P1 NEQ 0 THEN STACK(P1,2) = STACK(P1,2) - 1 $
        GO TO OPCHECK $
        END$
       END$
      END$
      POINT = P2 $
      P1 =  P2 = 0 $
      GO TO S2 $
  S1:
      POINT = POINT - 1 $
  S2:
      IF DUMMY(POINT) LSS 0 THEN
       BEGIN

  COMMENT IF VARIABLE INSERT 'V'AND NUMBER $
```

```
        P1 = P1 + 1 $
        BUFFER(P1) = -DUMMY(POINT) $
        P1 = P1 + 1 $
        BUFFER(P1) = 'V' $
        GO TO S4 $
        END$
        IF SYMTAB(DUMMY(POINT),1) EQL 0 THEN
         BEGIN

COMMENT IF CONSTANT OPERAND INSERT SYMBOLIC VALUE $

         C3 = DUMMY(POINT) $
         FOR C2 = SYMTAB(C3,5) STEP -1 UNTIL SYMTAB(C3,4) DO
         BEGIN
          P1 = P1 + 1 $
          BUFFER(P1) = SYMVALUE(C2) $
         END$
        GO TO S4 $
        END$
        P2 = P2 + 1 $
        STACK(P2,1) = DUMMY(POINT) $
        STACK(P2,2) = 0 $
S3:
    IF P2 EQL 2 THEN
    BEGIN

COMMENT INSERT RIGHT BRACKET IF CONDITIONS HOLD $

      IF (SYMTAB(STACK(P2,1),3) LSS SYMTAB(STACK(P2-1,1),3)) OR
       (SYMTAB(STACK(P2,1),3) EQL SYMTAB(STACK(P2-1,1),3) AND
       STACK(P2-1,2) EQL 0 ) THEN
        BEGIN
        P1 = P1 + 1 $
        BUFFER(P1) = ')' $
        END$
       END$
      GO TO S1 $
S4:
    IF P2 EQL 0 THEN GO TO SEND $
    STACK(P2,2) = STACK(P2,2) + 1 $
    IF STACK(P2,2) EQL 2 THEN GO TO S6 $
S5:
COMMENT IF A VARIABLE OPERATOR INSERT SOME DISTINCTIVE SYMBOL.$

    IF SYMTAB(STACK(P2,1),2) EQL 0 THEN
     BEGIN
     P1 = P1 + 1 $
     BUFFER(P1) = ' ' $
     P1 = P1 + 1 $
     BUFFER(P1) = SYMTAB(STACK(P2,1),1) $
     P1 = P1 + 1 $
     BUFFER(P1) = '$' $
     P1 = P1 + 1 $
     BUFFER(P1) = ' ' $
     END
    ELSE BEGIN
```

```
COMMENT IF A CONSTANT INSERT SYMBOLIC REPRESENTATION. $

   C3 = STACK(P2,1) $
   FOR C2 = SYMTAB(C3,5) STEP -1 UNTIL SYMTAB(C3,4) DO
    BEGIN
    P1 = P1 + 1 $
    BUFFER(P1) = SYMVALUE(C2) $
    END$
   END$
   IF SYMTAB(STACK(P2,1),1) GEQ 2 THEN GO TO S1 ELSE GO TO S7 $
S6:

COMMENT INSERT LEFT BRACKET IF STACK CONDITIONS TRUE $

   IF P2 EQL 2 THEN BEGIN
    IF (SYMTAB(STACK(P2,1),3) LSS SYMTAB(STACK(P2-1,1),3)) OR
     (SYMTAB(STACK(P2,1),3) EQL SYMTAB(STACK(P2-1,1),3) AND
      STACK(P2-1,2) EQL 0) THEN
      BEGIN
      P1 = P1 + 1 $
      BUFFER(P1) = '(' $
      END$
     END$
S7:
    P2 = P2 - 1 $
    GO TO S4 $
SEND:
   P2 = 1 $

COMMENT INVERT STRING AND PRINT IN 120 CHAR LINES $

   FOR C1 = P1 STEP -1 UNTIL 1 DO
    BEGIN
    BUFF2(P2) = BUFFER(C1) $
    P2 = P2 + 1 $
    IF P2 EQL 120 THEN
      BEGIN
      WRITE(F10,BUFF2) $
      P2 = 1 $
      END$
    END$
   WRITE(F10,BUFF2) $
   END$



   PROCEDURE APPLYOP(A,P1,OP,B,P2) $
   INTEGER ARRAY A,B $
   INTEGER P1,P2,OP $

COMMENT PROCEDURE APPLIES OPERATOR OP TO STRUCTURE A ROOTED AT
 P1 TO PRODUCE B ROOTED AT P2. $

  BEGIN
   INTEGER ARRAY VEC1,VEC2(-MAXSTAX:1) $
   INTEGER I,J,C1,C2,C3,C4,C5,C6,C7,TAG,D1,D2,VAR,PNT,N2 $
```

```
COMMENT SELECT OPERATOR AND POSITION OF APPLICATION $

   I = OP//FACTOR $
   J = MOD(OP,FACTOR) $
   D1 = RULESL(I,1) $
   D2 = RULE(D1,4) $
   TAG = 0 $

COMMENT INITIALISE VECTOR FOR VARIABLES IN A AND NOTE MINIMUM.
 VEC1,VEC2 KEEP TRACK OF VARIABLE NAME AND POSITION $

   FOR C1 = P1 STEP 1 UNTIL A(P1,4) DO
    IF A(C1,1) LSS 0 THEN
     BEGIN
      VEC2(A(C1,1)) = 0 $
      IF A(C1,1) LSS TAG THEN TAG = A(C1,1) $
      END$
    IF TAG LSS 0 THEN TAG = TAG - 1 $

COMMENT NOTE POINT AT WHICH OP IS TO BL APPLIED AND THE VALUES
 IN A WHICH REPLACE THE VARIABLES IN THE INPUT OF OP $

      FOR C1 = D1 STEP 1 UNTIL D2 DO
      IF RULE(C1,1) LSS 0 THEN VEC1(RULE(C1,1)) = 0 $
     PNT = P1+J $
   FOR  C1 = D1 STEP 1 UNTIL D2 DO
   IF RULE(C1,1) GEQ 0 THEN PNT = PNT + 1
   ELSE BEGIN
   VAR = RULE(C1,1) $
   IF VEC1(VAR) EQL 0 THEN VEC1(VAR) = PNT
  ELSE
   IF A(VEC1(VAR),1) LSS 0 THEN
   BEGIN
    C3 = A(VEC1(VAR),1) $
    IF A(PNT,1) GTR 0 THEN
     VEC2(C3 ) = VEC1(VAR) = PNT $
   END
   ELSE
    IF A(PNT,1) LSS 0 THEN
     VEC2(A(PNT,1)) = VEC1(VAR) $
    PNT = A(PNT,4) + 1 $
   END$
   J = P1 + J $
   C1 = P1 $
   C2 = P2 $
LOOP:
   IF C1 LEQ A(P1,4) THEN BEGIN
    IF C1 NEQ J THEN BEGIN

COMMENT IF WORKING WITH SECTION OF A OUTSIDE OPERATOR I.E. LT J
 OR GT A(J,4) INSERT VALUE DIRECTLY TO OUTPUT UNLESS IT IS A
  VARIABLE REQUIRING SUBSTITUTION - IF SO SELECT CORRECT SUBST
  VALUE $

     IF A(C1,1) GTR 0 OR VEC2(A(C1,1)) EQL 0 THEN
     BEGIN
```

```
        B(C2,1) = A(C1,1) $
        C2 = C2 + 1 $
        C1 = C1 + 1 $
       END
      ELSE BEGIN
      N2 = VEC2(A(C1,1)) $
      FOR C3 = N2 STEP 1 UNTIL A(N2,4) DO
       BEGIN
       B(C2,1) = A(C3,1) $ C2 = C2+1$ END$
       C1 = C1 + 1 $
      END
      END
       ELSE BEGIN

   COMMENT INSERT OUTPUT OF TO B - IF CONSTANT INSERT DIRECTLY ELSE
    FIND CORRECT SUBSTITUTION VALUE FOR VARIABLE $

       N2 = RULESR(I,1) $
       FOR C3 = N2 STEP 1 UNTIL RULE(N2,4) DO
        IF RULE(C3,1) GTR 0 THEN
        BEGIN
        B(C2,1) = RULE(C3,1) $ C2 = C2+1 $ END
          ELSE IF VEC1(RULE(C3,1)) EQL 0 THEN
           BEGIN

   COMMENT IF SIMPLY NEW VARIABLE THEN INSERT - TAG USED TO
    PREVENT CONFUSION WITH EXISTING VARIABLES $

        IF TAG EQL 0 THEN B(C2,1) = RULE(C3,1)
        ELSE B(C2,1) = TAG $
       C2 = C2 + 1 $
      END
      ELSE BEGIN
       C4 = VEC1(RULE(C3,1)) $
       FOR C5 = C4 STEP 1 UNTIL A(C4,4) DO
        IF A(C5,1) GTR 0 OR
         VEC2(A(C5,1)) EQL 0
          THEN BEGIN
          B(C2,1) = A(C5,1) $ C2 = C2+1 $ END
          ELSE BEGIN

   COMMENT CHECK FOR SUBSTITUTIONS WITHIN SUBSTITUTIONS AND
     INSERT CORRECT VALUE $

          C6 = VEC2(A(C5,1)) $
          FOR C7 = C6 STEP 1 UNTIL A(C6,4) DO
           BEGIN
            B(C2,1) = A(C7,1) $ C2 = C2+1 $ END$
       END$
      END$
      C1 = A(J,4) + 1 $
      END$
      GO TO LOOP $
      END$
      POSMAP(B,P2,C2-1) $

   COMMENT PLACE B IN CORRECT FORM FOR PROCESSING $
```

```
      END$



       PROCEDURE ANALYSERULE(NUMBER) $
        INTEGER NUMBER $

   COMMENT PROCEDURE ANALYSES AN OPERATOR TO DETERMINE THE PROBABLE
    EFFECT OF APPLYING IT. TWO TABLES(TERMTAB &VARTAB) RESPECTIVELY
     RECORD THE POSITION OF CONSTANT SYMBOLS IN THE OUTPUT STRUCTURE AND
     THE POSITION OF OUTPUT VARIABLES IN TERMS OF THEIR POSITION
     IN THE INPUT $

     BEGIN
      INTEGER ARRAY PACKSTACK(1:MAXSTAX,1:3),VTAB(1:MAXSTAX,1:2),
      LSTACK(1:MAXSTAX,1:2) $
      INTEGER LPOINT,PL,PR,C1,C2,C3,VCOUNT,TEMP,TAB,TEST $
      BOOLEAN FLAG1,FLAG2 $
      REAL F1,F2,D1,D2 $
      F1 = RULESL(NUMBER,2) - RULESL(NUMBER,1) + 1 $
      F2 = RULESR(NUMBER,2) - RULESR(NUMBER,1) + 1 $

   COMMENT SET INITIAL POINTERS AND CORRECT LINKS FOR INPUT AND
    OUTPUT STRUCTURES $

      VCOUNT = LPOINT = 0 $
      PL = RULESL(NUMBER,1) $
      PR = RULESR(NUMBER,1) $
      POSMAP(RULE,PL,RULESL(NUMBER,2)) $
      POSMAP(RULE,PR,RULESR(NUMBER,2)) $
      TAB = TERMTABENTRY(NUMBER) = TTBPNT + 1 $
      TEST = VATPNT + 1 $
   S1:
      IF PR LEQ RULESR(NUMBER,2) THEN
       BEGIN

   COMMENT IF WITHIN STRUCTURE INCREMENT MAIN TABLE POINTER $

      TTBPNT = TTBPNT + 1 $
      TERMTAB(TTBPNT,3) = 0 $
      IF LPOINT NEQ 0 THEN BEGIN

   COMMENT DECREASE TOP OPVALUE - PACKSTACK CONTROLS LINKS BETWEEN
    SUBSTRUCTURES $

       PACKSTACK(LPOINT,1) = PACKSTACK(LPOINT,1) - 1 $
       IF PACKSTACK(LPOINT,1) LSS PACKSTACK(LPOINT,3) THEN
        BEGIN
         C3 =TERMTAB(PACKSTACK(LPOINT,2),2) $
   S3:

   COMMENT IF AT CORRECT POSITION IN TLRMTAB INSERT FORWARD POINTER
    TO CURRENT SUBSTRUCTURE $

        IF TERMTAB(C3,3) EQL 0 THEN TERMTAB(C3,3) = TTBPNT
       ELSE BEGIN
```

```
        C3 = TERMTAB(C3,3) $ GO TO S3 $ ENDS
      ENDS
     END$
       IF RULE(PR,1) EQL RULE(PL,1) THEN
        BEGIN

COMMENT IF VALUE IS UNCHANGED INSERT ZERO TO TERMTAB ELSE INSERT
 VALUE $

        FLAG1 = FALSE $
        TERMTAB(TTRPNT,1) = 0 $
        END
        ELSE BEGIN
         FLAG1 = TRUE $
         TERMTAB(TTRPNT,1) = RULE(PR,1) $
        END$

 COMMENT SET VALUE TO 4TH ENTRY FOR CORRESPONDENCE CHECK $

     TERMTAB(TTRPNT,4) = RULE(PR,1) $

COMMENT IF VALUE IN OUTPUT IS OPERATOR THAT MATCHES INPUT
 CONSTANT INSERT TO PACKSTACK .INCREMENT FIRST SON . IF IT MATCHES
 A VARIABLE INSERT ZERO TO FIRST SON . $

       IF RULE(PR,5) GTR 0 THEN
        BEGIN
        IF RULE(PL,1) GTR 0 THEN
         BEGIN
         LPOINT = LPOINT + 1 $
         PACKSTACK(LPOINT,1) = RULE(PR,5) $
         PACKSTACK(LPOINT,2) = TTRPNT $
         PACKSTACK(LPOINT,3) = RULE(PR,5) - 1 $
         TERMTAB(TTRPNT,2) = TTRPNT + 1 $
        END
        ELSE BEGIN
         TERMTAB(TTRPNT,2) = 0 $  PR = RULE(PR,4) $ END
        END
       ELSE BEGIN

COMMENT SET FIRST SON TO ZERO. IF OUTPUT VALUE IS VARIABLE
 TEST IF IT HAS BEEN DEALT WITH EARLIER. $

        TERMTAB(TTRPNT,2) = 0 $
        IF RULE(PR,1) LSS 0 AND FLAG1 THEN
         BEGIN
         FOR C1= 1 STEP 1 UNTIL VCOUNT DO
          IF VTAB(C1,1) EQL RULE(PR,1) THEN
           BEGIN
           TERMTAB(TTRPNT,1) = VTAB(C1,2) $ GO TO S2 $
         END$
        FLAG2 = FALSE $

COMMENT SELECT MATCHING VARIABLE(S) IN INPUT - USE LSTACK TO
 FIND CORRECT BACKWARD LINKS $

         FOR C1 = RULESL(NUMBER,1) STEP 1 UNTIL RULESL(NUMBER,2) DO
```

```
          IF RULE(C1,1) EQL RULE(PR,1) THEN
          BEGIN
          C2 = 0 $
          TEMP = C1 $
          FOR C2 = C2 + 1 WHILE TEMP NEQ 0 DO
           BEGIN
           LSTACK(C2,1) = TEMP $
           LSTACK(C2,2) = RULE(TEMP,2) $
           TEMP = RULE(TEMP,3) $
          END$

COMMENT IF MORE THAN ONE VARIABLE SET FLAG IN VARTAB $

         IF FLAG2 THEN VARTAB(VATPNT,4) = -1
         ELSE BEGIN
          VCOUNT = VCOUNT + 1 $
          VTAB(VCOUNT,1) = RULE(PR,1) $
           VTAB(VCOUNT,2) = TERMTAB(TTBPNT,1) = -(VATPNT + 1) $
    END$
         FLAG2 = TRUE $

COMMENT INSERT POINTERS IN FORWARD ORDER TO VARTAB FROM
 LSTACK . POINTERS SHOW RELATION TO BASE OF INPUT .$

         IF C2 EQL 2 THEN
          BEGIN
          VATPNT = VATPNT + 1 $
          VARTAB(VATPNT,1) = VARTAB(VATPNT,2) = 0 $
        END
         ELSE FOR C3 = C2-1 STEP -1 UNTIL 2 DO
          BEGIN
           VATPNT = VATPNT + 1 $
           VARTAB(VATPNT,1) = RULE(LSTACK(C3,1),1) $
           VARTAB(VATPNT,2) = RULE(LSTACK(C3,1),5) $
           VARTAB(VATPNT,3) = LSTACK(C3-1,2) $
           VARTAB(VATPNT,4) = VATPNT + 1 $
        END$
        VARTAB(VATPNT,4) = 0 $

COMMENT INCREMENT POINTER FOR INPUT $

        PL = RULE(PL,4) $
       END$
        IF NOT FLAG2 THEN
         BEGIN

COMMENT IF VARIABLE ONLY EXISTS IN OUTPUT THEN INSERT VALUL
 TO TERMTAB AND SET FIRST SON TO - FOR INDICATOR $

         TERMTAB(TTBPNT,1) = RULE(PR,1) $
         TERMTAB(TTBPNT,2) = -1 $ ENDS
        END$
       END$
S2:

COMMENT RESET PACKSTACK TO LOWER LEVEL OF OP IF NECESSARY $
```

```
      IF LPOINT NEQ 0 THEN BEGIN
        IF PACKSTACK(LPOINT,1) EQL 0 THEN BEGIN
          LPOINT = LPOINT - 1$ GO TO S2 $ END$
         END$

   COMMENT INCREMENT POINTERS FOR INPUT AND OUTPUT $

        PL = PL + 1 $
        PR = PR + 1 $
        GO TO S1 $
        END$
        D1 = 0.0 $
        FOR C1 = TAB STEP 1 UNTIL TTBPNT DO
         IF TERMTAB(C1,1) NEQ 0 THEN D1 = D1 + 1.0 $
        D2 = VATPNT - TEST + 1   $

   COMMENT COMPUTE VALUE TO REFLECT COMPLEXITY OF OP $

     RCOMP(NUMBER) = RCFACTOR1/(RCFACTOR2*D1*RCFACTOR3*ABS(F1-F2))
        +RCFACTOR4/(F1+F2) +1.0/(RCFACTOR5+D2) $
        END$



        PROCEDURE CLEARUP $

   COMMENT PROCEDURE RESETS ALL ARRAYS TO ZERO AND RE-INITIALISES
    THE POINTERS $

       BEGIN
        INTEGER C1,C2,C3 $
        FOR C1=(1,1,MAXFLT) DO
        FOR C2=(1,1,5) DO  LIT(C1,C2) = 0 $
        FOR C1=(1,1,MAXGOALS) DO GOALS(C1) = 0 $
        FOR C1=(1,1,MAXGL) DO
         FOR C2 =(1,1,2) DO GOALIST(C1,C2) = 0 $
        FOR C1 = (1,1,MAXNODES) DO BEGIN
        NODEVAL(C1) = 0.0 $
        FOR C2 =(1,1,10) DO NODL(C1,C2) = 0 $
        END$
        FOR C1 = (1,1,MAXOPS) DO
         OPER(C1,4) = C1+1$
        FREEOPS = 1 $
        LASTOP = MAXOPS $
        FOR C1=(1,1,MAXSUBOPS) DO SUBOPLIST(C1) = 0 $
        NEXTSUBOP = 1 $
        NEXTELT = 2 $
        NEXTGOAL = 0 $
        TOPGL = 0 $
       END$



        PROCEDURE RESULTPRINT(LBF,PENT,PENL) $
        INTEGER LBF $
        REAL PENT,PENL $
```

```
COMMENT PROCEDURE DETERMINES PENETRANCE AND EFFECTIVE BRANCHING
  FACTOR OF SOLUTION $

    BEGIN
    REAL ARRAY X(0:27) $
    INTEGER  G1,G2 $
    WRITE(' ') $
    PENL = PENL/PENT $
    WRITE('PENETRANCE',PENL) $
    WRITE(' ') $
    IF LBF EQL 1 OR NEXTNODE GTR 300 THEN
     WRITE('NO EBF CALCULATED')
    ELSE BEGIN
     IF LBF EQL (NEXTNODE-1) THEN
      WRITE('EFFECTIVE BRANCHING FACTOR:  1.0')
       ELSE BEGIN
       POSITION(FILE('A',0)) $
       G1 = G2 = (LBF-2)*300 +(NEXTNODE-2) $
       READ(FILE('A',G2),X) $
       G1 = MOD(G1,28) $
       IF X(G1) LSS ERR THEN WRITE('NO EBF ALCULATED')
      ELSE WRITE('EFFECTIVE BRANCHING FACTOR:',X(G1)) $
END$
END$
END$



     PROCEDURE SOLVER2 $

COMMENT PROCEDURE CONTROLS OPERATION OF SDPS ALGORITHM.
 IF SOLUTION IS OBTAINED WITHIN TIME LIMIT IT IS PRINTED
  ELSE A FAILURE MESSAGE IS GIVEN. $

   BEGIN
    INTEGER ARRAY NEXTA,NEXTB,TOPGOAL(1:MAXSTRING,1:5) $
    STRING PPR(30) $
    REAL PENL,PENT $
    INTEGER  C1,C2,BESTNODE,SOLTIME,OP1,NAME1,NAME2,OPLEVEL,S1,S2,
     NEWOP,P1,P2,I,J,CURRENTDEPTH,CURRENTLEVEL,G1,G2,FATHER,EMT,
     HEAD,PUT,TOPGOLNAME,NEXT,NEXTOP,N,LAST,C3,C4,NEXTBEST,
     LBF,TBF $
    BOOLEAN NEWA,NEWB $
    LIST LIST2('APPLY OP',G1) $
    FORMAT F10(A0,1,J60,S8,13) $
    FORMAT F11(S30,A1,1) $
    FORMAT F12(J10,S10,A1,1) $
    FORMAT F13(J10,S17,A1,1) $

COMMENT INITIALISE FIRST NODE BY ESTABLISHING
 CANONICAL NAMES OF INITIAL STRUCTURES AND SETTING
 PARAMETERS FOR DEPTH,SUBGOAL LEVEL,ETC.
  EVALUATE THE FIRST NODE. $

    FOR C1=(1,1,30) DO PPR(C1) = '-' $
    C1 =C2 = 1 $
    G1 = G2 = 1 $
```

```
      PENL = PENT = 0.0 $
      POSMAP(STRA,C1,LENGA) $
      POSMAP(STRB,C2,LENGB) $
      WRITE(F11,PPR) $
      WRITE(F12,'PROVE THAT') $
      POLISHINFIX(STRA,G1) $
      WRITE(F13,'IS EQUIVALENT TO') $
      POLISHINFIX(STRB,G2) $
      WRITE(F11,PPR) $
      WRITE(' ') $
      IF NAMELT(STRA,NAME1) THEN ERRORWRITE(6)
       ELSE IF NAMELT(STRB,NAME2) THEN ERRORWRITE(6) $
      COPY(STRB,C1,TOPGOAL,C1) $
      TOPGOLNAME = NAME2 $
      CURRENTDEPTH = 1 $
      MAXTIME = MAXTIME*10000 $
      NEWGOAL(NAME1,NAME2,CURRENTDEPTH) $
      NODE(1,1) = 0 $
      NODE(1,2) = NAME1*FACTOR + NAME2 $
      BESTNODE = 1 $
      NEXTNODE = 2 $
      NODE(1,7) = NODE(1,9) = 0 $
      NODE(1,10) = 1 $
      NODE(1,5) = MAXSUBGOALS $
      OPLEVEL = OPDLEVEL $
      DIFFNUM = 0 $
      ZERODIFFS(STRA,STRB,G1,G2) $
      IF DIFFNUM EQL 0 THEN GO TO SUCCESS $
      SOLTIME = 0 $
      N = TIME $
      OPDIFFGENERATE(STRA,STRB,C1,C2,OPLEVEL,MAXSUBGOALS,1) $
      IF OPNUM EQL 0 THEN GO TO FAIL1 $
      ORDEROPS $
      INSERTOPS(BESTNODE) $
      EVALUATE(BESTNODE) $
   ST1:

    COMMENT IF NO MORE NODES OR TIME THEN ADMIT FAILURE. $

      SOLTIME = SOLTIME + TIME $
      IF SOLTIME GTR MAXTIME THEN GO TO FAIL2 $
      IF BESTNODE EQL 0 THEN GO TO FAIL1 $

   COMMENT SELECT NEXT OPERATOR AT BEST NODE . RE-EVALUATE
    THE NODE AND ESTABLISH ITS RELATION TO REST OF TREE$

      UNPACK2(NODE(BESTNODE,4),N,OP1) $
      PACK2(N-1,OPER(OP1,4),NODE(BESTNODE,4)) $
      OPER(LASTOP,4) = OP1 $
      LASTOP = OP1 $
      EVALUATE(BESTNODE) $
      BACKONE(BESTNODE,NEXTBEST) $
      S1 = NODE(BESTNODE,2)//FACTOR $
      S2 = MOD(NODE(BESTNODE,2),FACTOR) $
      CURRENTDEPTH = NODE(BESTNODE,10) +1 $
      CURRENTLEVEL = NODE(BESTNODE,5) $
      IF OPER(OP1,3) EQL 1 THEN
```

```
            BEGIN

      COMMENT IF OP CAN BE APPLIED THEN GENERATE NEW STRUCTURE
       BY APPLYING IT TO RETRIEVED OBJECT. $

            RETRIEVELT(S1,STRA,LENGA) $
            POSMAP(STRA,G1,LENGA) $
            NEWOP = OPER(OP1,1) $
            NEWB = FALSE $
            APPLYOP(STRA,G1,NEWOP,NEXTA,G2) $
            PENT =PENT + 1.0 $

      COMMENT DETERMINE IF STRUCTURE IS NEW OR OLD $

           IF NAMELT(NEXTA,NAME1) THEN NEWA = FALSE
           ELSE NEWA = TRUE $
            RETRIEVELT(S2,NEXTB,LENGB) $
            NAME2 = S2 $
            POSMAP(NEXTB,G1,LENGB) $
          END
           ELSE BEGIN

      COMMENT IF OPERATOR NOT APPLICABLE THEN SET UP SUBGOAL
       TO ATTAIN STATE IN WHICH IT MAY BE APPLIED.
       IF LEVEL OF THIS SUBGOAL IS ABOVE MAXIMUM THEN
        SELECT NEXT BEST NODE $
                                                  *
           CURRENTLEVEL = CURRENTLEVEL - 1 $
           IF CURRENTLEVEL EQL 0 THEN
            GO TO GOBACK1 $
           P1 = 1 $
           RETRIEVELT(S1,NEXTA,LENGA) $
           NAME1 = S1 $
           POSMAP(NEXTA,P1,LENGA) $
           NEWA = FALSE $
           EMT = OPER(OP1,2) $
           NEWOP = -OPER(OP1,1) $
           I = OPER(OP1,1)//FACTOR $
           J = MOD(OPER(OP1,1),FACTOR) $
           P2 = 1 $
           C1 = J + 1 $
           COPY(RULE,RULESL(I,1),NEXTB,P2) $
           LENGB = RULESL(I,2) - RULESL(I,1) + P2 $
           BUILDGOAL(NEXTA,P1,C1,STRB,P2,NEXTB,LENGB) $

      COMMENT DETERMINE IF SUBGOAL IS NEW STRUCTURE. $

           POSMAP(NEXTB,P2,LENGB) $
           IF NAMELT(NEXTB,NAME2) THEN NEWB = FALSE
           ELSE NEWB = TRUE $
           END.
      ST2:

      COMMENT IF EITHER STRUCTURE IS NEW FILE THE NODE ELSE
       DETERMINE IF THIS COMBINATION HAS OCCURRED BEFORE$

           IF NEWA THEN
```

```
        NEWGOAL(NAME1,NAME2,NEXTNODE)
       ELSE BEGIN
        IF (NOT NEWA) AND NEWB THEN
        INSERTGOAL(NAME1,NAME2,NEXTNODE)
        ELSE BEGIN

 COMMENT IF THIS IS AN OLD COMBINATION ESTABLISH
  WHETHER A SHORTER CORRECT PATH HAS BEEN FOUND. IF
   IF SO TRANSFER OLD NODE TO NEW POSITION ,IN EITHER
   CASE CYCLING IS PREVENTED. $

        N = NEXTNODE $
        IF TESTGOAL(NAME1,NAME2,N) THEN BEGIN
         IF CURRENTDEPTH LSS NODE(N,10) THEN BEGIN
          FATHER = NODE(N,1) $
         HEAD = PNT = NODE(FATHER,8) $
         LAST = 0 $
 ST3:
        IF PNT EQL N THEN
         BEGIN
         IF LAST EQL 0 THEN
         NODE(FATHER,8) = NODE(HEAD,9)
        ELSE NODE(LAST,9) = NODE(PNT,9) $ END
         ELSE BEGIN
           LAST = PNT $
           PNT = NODE(PNT,9) $
           IF PNT EQL 0 THEN ERRORWRITE(8) $
           GO TO ST3 $
        END$
        NEXT = PNT = NODE(FATHER,8) $
        IF PNT NEQ 0 AND NODE(FATHER,3) EQL N THEN
         BEGIN
         FOR NEXT = NODE(NEXT,9) WHILE NEXT NEQ 0 DO
          IF NODEVAL(NEXT) GTR NODEVAL(PNT) THEN PNT = NEXT $
       IF NODEVAL(PNT) GTR NODEVAL(FATHER) THEN
        NODE(FATHER,3) = PNT
         ELSE NODE(FATHER,3) =  FATHER $
       BACKONE(PNT,PNT) $
       END$
  COMMENT

  IF NODE IS SWITCHED LINK IT IN AND RECONFIGURE THE TREE. $

       LINK(BESTNODE,N) $
       BACKUP(N,BESTNODE) $
       GO TO ST1 $
       END
        ELSE GO TO GOBACK1 $
       END$
 END$
 END$

 COMMENT GENERATE SET OF ZEROLEVEL DIFFERENCES. $

       DIFFNUM = 0 $
       ZERODIFFS(NEXTA,NEXTB,G1,G2) $
       IF DIFFNUM EQL 0 THEN
```

```
            BEGIN

      COMMENT IF NO DIFFERENCES DETERMINE WHETHER MAIN PROBLEM SOLVED$

          IF CURRENTLEVEL EQL MAXSUBGOALS OR NAME2 EQL TOPGOLNAME THEN
            GO TO SUCCESS
           ELSE BEGIN

      COMMENT IF A SUBGOAL HAS BEEN SOLVED FILE THE NODE. DETERMINE
        OPERATOR WHICH GENERATED SUBGOAL AND APPLY IT. $

            LINK(BESTNODE,NEXTNODE) $
            C1 = 0 $
            PNT = NEXTNODE $
            NODE(PNT,7) = NEWOP $
          FILENODE(NAME1,NAME2,CURRENTLEVEL,C1,C1,CURRENTDEPTH) $
          NODE(PNT,4)  = 0 $
          NODE(PNT,3) = PNT $
          NODEVAL(PNT) = 0 $
          CURRENTDEPTH = CURRENTDEPTH + 1 $
          FATHER = BESTNODE $
      ST4:
          IF NODE(FATHER,5) EQL NODE(PNT,5) AND
          SUBGOAL(FATHER) THEN OP1 = NODE(FATHER,7)
          ELSE BEGIN
            FATHER = NODE(FATHER,1) $ GO TO ST4 $ END$
        UNPACKOP(OP1,NEXTA,P1,FATHER) $
        NEWOP = OP1 $

      COMMENT RETRIEVE SUBGOAL VALID BEFORE THIS SUBGOAL STARTED. $

          FATHER = NODE(FATHER,1) $
          NAME2 = MOD(NODE(FATHER,2),FACTOR) $
          RETRIEVELT(NAME2,NEXTB,LENGB) $
          POSNAP(NEXTB,P2,LENGB) $
          NEWB = FALSE $
          COPY(NEXTA,P1,STRA,P1) $

      COMMENT GENERATE NEW STRUCTURE BY APPLYING OP. DETERMINE
       WHETHER RESULT IS NEW OR OLD AND RESET SUBGOAL LEVEL. $

          APPLYOP(STRA,P1,OP1,NEXTA,P2) $
          PENT = PENT + 1.0 $
          IF NAMELT(NEXTA,NAME1) THEN
           NEWA = FALSE ELSE NEWA = TRUE $
          CURRENTLEVEL = NODE(FATHER,5) $
          BESTNODE = PNT $

      COMMENT RETURN TO TEST NEW NODE FOR CYCLING. $

          GO TO ST2 $
          END$ END$

      COMMENT GENERATE SET OF OPERATORS RELEVENT TO DIFFERENCES
          IF NONE GO TO SELECT NEXT BEST NODE FOR EXPANSION. $

          IF NEWOP LSS 0 THEN PNT = CURRENTLEVEL+1
```

```
      ELSE PNT = CURRENTLEVEL $
     OPDIFFGENERATE(NEXTA,NEXTB,G1,G2,OPLEVEL,PNT,BESTNODE) $
     IF OPNUM EQL 0 THEN GO TO GOBACK1 $

  COMMENT ORDER OPERATORS AND ATTACH TO NODE ,
   FILE AND LINK THE NODE , $

     ORDEROPS $
     INSERTOPS(NEXTNODE) $
     LINK(BESTNODE,NEXTNODE) $
     N = NEXTNODE $
     IF NEWOP LSS 0 THEN BEGIN PACKSUBOP(-NEWOP,NEXTA,P1,NEXTNODE)$
           PACKELT(EMT,-NEWOP,NEXTA,P1,NEXTNODE) $ END
     ELSE NODE(NEXTNODE,7) = NEWOP $
     FILENODE(NAME1,NAME2,CURRENTLEVEL,C1,C1,CURRENTDEPTH) $

 COMMENT EVALUATE THE NODE AND SELECT THE BEST NODE FOR
     EXPANSION, RETURN TO START OF CYCLE, $

     EVALUATE(N) $
     BACKUP(N,BESTNODE) $
     GO TO ST1 $
 GOBACK1:
   BESTNODE = NEXTBEST $
     GO TO ST1 $
 FAIL1:

 COMMENT ADMIT FAILURE DUE TO EXCEEDING MAXTIME OR
     NO NODES LEFT TO EXPAND $

     WRITE('NO SOLUTION FOUND') $
     GO TO SOLVEND $
 FAIL2:
     WRITE('MAXTIME EXCEEDED - SECONDS') $
     SOLTIME = SOLTIME//10000 $
     WRITE(SOLTIME) $
     GO TO SOLVEND $
 SUCCESS:

 COMMENT OUTPUT SOLUTION WITH MEASURES OF EFFICIENCY$

     WRITE(' ') $
     SOLTIME = SOLTIME//10000 $
     WRITE('SOLUTION TIME(SECS):',SOLTIME) $
     FATHER = NEXTNODE $
     LINK(BESTNODE,NEXTNODE) $
     NODE(NEXTNODE,7) = NEWOP $
     FILENODE(NAME1,NAME2,CURRENTLEVEL,C1,C1,CURRENTDEPTH) $
     C1 = 1 $
     OPLIST(C1,1) = FATHER $
     NAME2 = NODE(FATHER,2)//FACTOR $
     LBF = 1 $
     FOR FATHER = NODE(FATHER,1) WHILE FATHER NEQ 0 DO BEGIN
      LBF = LBF + 1 $
      IF NOT SUBGOAL(FATHER) THEN BEGIN
     NAME1 = NODE(FATHER,2)//FACTOR $
     IF NAME1 EQL NAME2 THEN
```

```
        BEGIN
        C1 = 0 $
        PENL = 0 $
        LBF = 1 $
       END$
          C1 = C1 + 1 $
          PENL = PENL + 1.0 $
          OPLIST(C1,1) = FATHER $
          END$
          END$
      RESULTPRINT(LBF,PENT,PENL) $
      P1 = 1 $
         FOR C2 =(C1,-1,1) DO
         BEGIN
          NAME2 = NODE(OPLIST(C2,1),2)//FACTOR $
          G1 =(NODE(OPLIST(C2,1),7))//FACTOR $
          WRITE(F10,LIST2) $
          RETRIEVELT(NAME2,STRB,LENGB) $
          POSMAP(STRB,P1,LENGB) $
          POLISHINFIX(STRB,P1) $
        END$
    SOLVEND:
      END$






       PROCEDURE INPUTDATA $

    COMMENT PROCEDURE BUILDS THE SYMBOL TABLE FOR THE CONSTANTS, IT ALSO
     PLACES THE OPERATORS IN THEIR CORRECT STRUCTURES AND READS THE PROBLEMS

       BEGIN
        INTEGER ARRAY LINK(1:30,1:3),COUNT(0:9) $
        STRING COMMAND(30),INPUT(80),INTVAL(10) $
        INTEGER C1,C2,C3,I,NEXTSYM,SYMPOS,DEGREE,RULEPUT,NEXT $
        BOOLEAN ENDOFCARD,INVALID,LEFT $
        LIST INP1(FOR C1=(1,1,30) DO FOR C2=(1,1,3) DO LINK(C1,C2)) $
       FORMAT FN1(A1,3I4) $
       FORMAT FN2(A1,S30) $
       FORMAT FN3(A1,S80) $
       SWITCH CONTYPE = T1,T3,T7,T14,T20 $



       PROCEDURE SELECTCOMMAND(POS,I) $
       INTEGER POS,I $

    COMMENT PROCEDURE DETERMINES WHETHER A COMMAND IS DEFINING CONSTANTS,
      OPERATORS OR PROBLEMS BY MATCHING AGAINST A TREE OF PREDEFINED
       SYMBOLS $

       BEGIN
        INTEGER P1,COMPUT $
```

```
        I = 0 $
        COMPNT = I $
        FOR P1=(2,1,POS) DO
         BEGIN
         IF INPUT(P1) EQL COMMAND(COMPNT) THEN
        BEGIN
         IF LINK(COMPNT,2) EQL 0 THEN BEGIN
          IF P1 NEQ POS THEN GO TO F2 $  END
         ELSE COMPNT = LINK(COMPNT,2) $
        END
        ELSE BEGIN
         IF LINK(COMPNT,1) EQL 0 THEN GO TO F2
          ELSE BEGIN
           COMPNT = LINK(COMPNT,1) $
           P1 = P1-1 $
          END$
         END$
         END$
        I = LINK(CONPNT,3) $
F2:
     END$


     INTEGER PROCEDURE CLASS(POS1,POS2)$
     INTEGER POS1,POS2 $

COMMENT PROCEDURE TRANSLATES SYMBOLIC TO INTEGER $

     BEGIN
      INTEGER C1,C2,TOT $
      TOT = U $
      FOR C1=(POS2+1,1,POS1) DO BEGIN
       FOR C2 =(1,1,10) DO
       IF INPUT(C1) EQL INTVAL(C2) THEN GO TO T1 $
       ERRORWRITE(5) $
T1:     TOT = TOT*10+COUNT(C2-1) $
     END$
     CLASS = TOT $
      END$


     INTEGER PROCEDURE PRECEDENCE(DEG) $
     INTEGER DEG $

COMMENT PROCEDURE DETERMINES PRECEDENCE OF CONSTANTS FOR CORRECT
   OUTPUT FORMAT $

     BEGIN
     IF DEG EQL 2 THEN PRECEDENCE = 1
      ELSE IF DEG EQL 1 THEN PRECEDENCE = 2
       ELSE PRECEDENCE = 0 $
     END$



     INTEGER PROCEDURE TABVALUE(P1,P2) $
     INTEGER P1,P2 $
```

```
       COMMENT PROCEDURE IDENTIFIES THE INDEX OF A SYMBOLIC CONSTANT IN THE
         SYMBOL TABLE $

          BEGIN
           INTEGER C1,C2,PNT,DIFF $
           DIFF = P2-P1 $
           IF INPUT(P1) EQL '#' THEN BEGIN
            TABVALUE= - CLASS(P2,P1) $    GO TO EXIT $
           END
          ELSE FOR C1=(1,1,NEXTSYM-1) DO
           BEGIN
           IF DIFF EQL (SYMTAB(C1,5)-SYMTAB(C1,4)) THEN
             BEGIN
             PNT = P1 $
              FOR C2 =(SYMTAB(C1,4),1,SYMTAB(C1,5)) DO
               IF SYMVALUE(C2) NEQ INPUT(PNT) THEN GO TO NEXT
                 ELSE PNT = PNT + 1 $
               TABVALUE = C1 $
               GO TO EXIT $
             END$
       NEXT:
          END$
          ERRORWRITE(5) $
       EXIT:
          END$


          BEGIN
          NEXTSYM = SYMPOS = 1 $
          INTVAL(1,10) ='0123456789' $
          FOR C1=(0,1,9) DO COUNT(C1) = C1 $
          READ(FN1,INP1) $
          READ(FN2,COMMAND) $
       T1:

       COMMENT JUMP ON COMMAND IDENTIFIER $

          READ(FN3,INPUT) $
          IF INPUT(1) EQL '%' THEN
           FOR C1 =(2,1,80) DO
             IF INPUT(C1) EQL ' ' THEN GO TO T2 $
          ERRORWRITE(5) $
       T2:
          C1=C1-1 $
          SELECTCOMMAND(C1,I) $
          GO TO COMTYPE(I) $
          ERRORWRITE(5) $
       T3:

       COMMENT IF A SET OF CONSTANT DEFINITIONS DETERMINE THE DEGREE AND PLACE
         EACH CONSTANT IN THE SYMBOL TABLE WITH ITS DEGREE,CLASS AND PRECEDENC
          AS WELL AS POINTERS TO THE DICTIONARY $

          READ(DEGREE) $
       T4:
          READ(FN3,INPUT) $
```

```
      C2 = -1 $
      FOR C1 = C2+2 WHILE C1 LEQ 80 DO
       BEGIN
       FOR C2=(C1,1,80) DO BEGIN
        IF INPUT(C2) EQL 'S' THEN GO TO T1 $
        IF INPUT(C2) EQL ',' THEN GO TO T5 $
       END$
       GO TO T4 $
   T5:
      C2=C2-1 $
      FOR C3=(C1,1,C2) DO
       IF INPUT(C3) EQL ':' THEN GO TO T6 $
       ERRORWRITE(S) $
   T6:
      SYMTAB(NEXTSYM,1) = DEGREE $
      SYMTAB(NEXTSYM,2) = CLASS(C2,C3) $
      SYMTAB(NEXTSYM,3) = PRECEDENCE(DEGREE) $
      SYMTAB(NEXTSYM,4) = SYMPOS $
      SYMTAB(NEXTSYM,5) = SYMPOS +C3-C1-1 $
      NEXTSYM = NEXTSYM + 1 $
      SYMVALUE(SYMPOS,SYMPOS+C3-C1-1) = INPUT(C1,C3-1) $
      SYMPOS = SYMPOS + C3-C1 $
      END$
      GO TO T4 $
   T7:

   COMMENT IF THE COMMAND DEFINES A SET OF OPERATORS DETERMINE THE NUMBER
    AND FOR EACH READ THE INPUT AND OUTPUT STRUCTURES,SETTING THE
     CORRECT POINTERS TO EACH STRUCTURE $

      RULEPNT = 1 $
      READ(RULENO) $
      FOR C1 = (1,1,RULENO) DO
       BEGIN
       RULESL(C1,1) = RULEPNT $
   T8:
       READ(FN3,INPUT) $
       ENDOFCARD = FALSE $
       C2 = 1 $
   T9:
        FOR C3 = (C2,1,80) DO BEGIN
         IF INPUT(C3) EQL ' ' THEN GO TO T10 $
           IF INPUT(C3) EQL ':' THEN GO TO T11 $
            IF INPUT(C3) EQL ';' THEN GO TO T12 $
             IF INPUT(C3) EQL 'S' THEN GO TO T14 $
        END$
        ENDOFCARD = TRUE $
   T10:
        RULE(RULEPNT,1) = TABVALUE(C2,C3-1) $
        IF INVALID THEN ERRORWRITE(5) $
        RULEPNT = RULEPNT + 1 $
        C2 = C3 + 1 $
         IF ENDOFCARD THEN GO TO T8 ELSE GO TO T9 $
   T11:
        RULE(RULEPNT,1) = TABVALUE(C2,C3-1) $
        IF INVALID THEN ERRORWRITE(5) $
        RULESL(C1,2) = RULEPNT $
```

```
              RULEPNT = RULEPNT + 1 $
              RULESR(C1,1) = RULEPNT $
              GO TO T3 $
         T12:
              RULE(RULEPNT,1) = TARVALUE(C2,C3-1) $
              IF INVALID THEN ERRORWRITE(5) $
              RULESR(C1,2) = RULEPNT $
              RULEPNT = RULEPNT+ 1 $
         T13:
            END$
            GO TO T1 $
         T14:

         COMMENT COMES HERE WHEN COMMAND DEFINES  PROBLEM INPUT $

            NEXT = 0 $
            LEFT = TRUE $
         T15:
            READ(FM3,INPUT) $
            ENDOFCARD = FALSE $
            C1 = 1 $
         T16:
             FOR C2 = (C1,1,80) DO BEGIN
              IF INPUT(C2) EQL ' ' THEN GO TO T17 $
               IF INPUT(C2) EQL ':' THEN GO TO T18 $
                IF INPUT(C2) EQL ';' THEN GO TO T19 $
              END$                            *
             ENDOFCARD = TRUE $
         T17:
             NEXT = NEXT + 1 $
             IF LEFT THEN STRA(NEXT,1) = TARVALUE(C1,C2-1)
               ELSE STRB(NEXT,1)  = TABVALUE(C1,C2-1) $
             IF INVALID THEN ERRORWRITE(5) $
             C1 = C2+1 $
             IF ENDOFCARD THEN GO TO T15 ELSE GO TO T16 $
         T18:
             NEXT = NEXT+1 $
             STRA(NEXT,1) = TARVALUE(C1,C2-1) $
             IF INVALID THEN ERRORWRITE(5) $
             LEFT = FALSE $
             LENGA = NEXT $
             NEXT = 0 $
             GO TO T15 $
         T19:
             NEXT = NEXT + 1 $
             STRB(NEXT,1) = TARVALUE(C1,C2-1) $
             IF INVALID THEN ERRORWRITE(5) $
             LENGB = NEXT $
             GO TO T1 $
         T20:
            END$
            END$



            BOOLEAN PROCEDURE SURGOAL(N) $
            INTEGER N $
```

```
      COMMENT PROCEDURE ESTABLISHES WHETHER A NODE IS A SUBGOAL
     BEGIN
       IF NODE(N,7) GTR 2**23 THEN SUBGOAL = TRUE
        ELSE SUBGOAL = FALSE $
      END$
     BEGIN
     TTBPNT = VATPNT = 1 $
     FACTOR = 32768 $
   COMMENT SET UP OPERATORS AND PROBLEM    $
      INPUTDATA $
      FOR C1=(1,1,RULENO) DO ANALYSERULE(C1) $
      FOR C1 = 1 STEP 1 UNTIL RULENO DO
       BEGIN
       WRITE('OPERATOR',C1) $
       WRITE(' ') $
       POLISHINFIX(RULE,RULESL(C1,1)) $
       POLISHINFIX(RULE,RULESR(C1,1)) $
       WRITE(' ') $
       END$
     FOR C1 = 1 STEP 1 UNTIL MAXOPS DO
      OPER(C1,4) = C1 + 1 $
      FREEOPS = 1 $
      LASTOP = MAXOPS $
   COMMENT START PROBLEM SOLVING PROCEDURE $
      SOLVER2 $
      END$
      END$
     COMMENT INITIALISE ALL PARAMETERS $
      READ(FMAX,MAXVALUES) $
      READ(FMAX,MAXVALUES2) $
      READ(ERR) $
      READ(FEVAL,EVALTYPE) $
      READ(FDEPTH,DEPTHTYPE) $
      READ(COMPBIAS) $
      READ(SPECBIAS) $
      READ(FDIFF,DIFFTYPE) $
      READ(FRC,RCTYPE) $
      READ(LENGTHBIAS) $
      NEXTSUBOP = 1 $
      NEXTELT = 1 $
      NEXTGOAL = 0 $
      TOPGL = 0 $
     MAIN1 $
```

```
MAINEND:
 END$
```

@FIN