A HEURISTIC PROGRAM FOR
SOLVING PROBLEMS STATED AS
NONDETERMINISTIC PROCEDURES

BY

RICHARD EARL FIRES

COMPUTER SCIENCE DEPARTMENT
CARNEGIE-MELLON UNIVERSITY
PITTSBURGH, PENNSYLVANIA
NOVEMBER 1968

SUBMITTED TO THE CARNEGIE-MELLON UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

THIS WORK WAS SUPPORTED BY THE ADVANCED RESEARCH
PROJECTS AGENCY OF THE OFFICE OF THE SECRETARY OF
DEFENSE (F 44620-67-C-0058) AND IS MONITORED BY THE AIR
FORCE OFFICE OF SCIENTIFIC RESEARCH. THIS DOCUMENT HAS
BEEN APPROVED FOR PUBLIC RELEASE AND SALE; ITS
DISTRIBUTION IS UNLIMITED.
ABSTRACT

WE DESCRIBE AN EFFORT TO DESIGN A HEURISTIC PROBLEM SOLVING PROGRAM IN WHICH THE PRIMARY CONCERNS ARE WITH THE GENERALITY OF THE PROGRAM'S INPUT LANGUAGE AND THE EFFECTIVENESS AND GENERALITY OF THE PROGRAM'S PROBLEM SOLVING METHODS.

TO OBTAIN THE DESIRED GENERALITY AND EASE OF PROBLEM STATEMENT IN AN INPUT LANGUAGE WE PROPOSE EXTENDING A PROGRAMMING LANGUAGE TO FORM A NONDETERMINISTIC LANGUAGE WHICH IS SUITABLE FOR STATING PROBLEMS. THE EXTENSIONS PRESERVE THE REPRESENTATIONAL POWER AND GENERALITY OF THE DATA AND CONTROL STRUCTURES OF THE BASE PROGRAMMING LANGUAGE. WE DISCUSS THE SCOPE AND LIMITATIONS OF NONDETERMINISTIC PROGRAMMING LANGUAGES AS INPUT LANGUAGES AND COMPARE THEM WITH THE INPUT LANGUAGES USED BY OTHER PROBLEM SOLVING PROGAMS.

THE PROGRAM WAS DESIGNED TO ACCEPT PROBLEMS STATED IN A PARTICULAR NONDETERMINISTIC PROGRAMMING LANGUAGE AND TO DEAL EFFECTIVELY WITH THE DIVERSITY OF PROBLEMS EXPRESSIBLE IN THE LANGUAGE. THE PROGRAM TRANSLATES A NONDETERMINISTIC PROCEUDRE INTO A HEURISTIC SEARCH PROBLEM IN WHICH EACH OBJECT IN THE SEARCH SPACE IS ITSELF A CONSTRAINT SATISFACTION PROBLEM. THE PROGRAM COMBINES HEURISTIC SEARCH METHODS AND CONSTRAINT SATISFACTION METHODS TO CONDUCT THE SEARCH AND TO SOLVE THE PROBLEMS DEFINED BY THE OBJECTS IN THE SPACE. WE PRESENT DISCUSSIONS OF THE PROGRAM'S PROBLEM SOLVING METHODS WITH EMPHASIS ON THEIR GENERALITY AND EXTENSIBILITY. THE BEHAVIOR OF THE PROGRAM ON A SET OF EXAMPLE PROBLEMS IS DESCRIBED AND ANALYZED.
ACKNOWLEDGEMENTS

FIRST OF ALL, I WISH TO THANK PROFESSOR ALLEN NEWELL FOR HIS STIMULATING AND THOUGHTFUL GUIDANCE OF THIS WORK. HIS SUPPORT HAS CONTINUALLY PROVIDED INSIGHT AND INSPIRATION THROUGHOUT THE PROJECT. I ALSO WISH TO THANK THE MANY PEOPLE WHO HELPED DEVELOP AND DEBUG THE INVALUABLE OS/360 RJE AND IPL-V SYSTEMS. FINALLY, I WISH TO THANK MY WIFE, ALICE, FOR TYPING AND ALTERING THE TEXT FILES THROUGH MANY DRAFTS OF THIS DOCUMENT AND FOR PROVIDING THE PERSONAL SUPPORT WHICH MADE THE WHOLE PROJECT POSSIBLE.
TABLE OF CONTENTS

I. INTRODUCTION 1

II. INPUT LANGUAGE FOR A GENERAL PROBLEM SOLVER 3
   A. THE PROBLEM OF DESIGNING AN INPUT LANGUAGE 3
   B. AN INPUT LANGUAGE PROPOSAL 9
   C. THE ARF INPUT LANGUAGE 11
   D. STATING PROBLEMS IN REF 18
      1. BOOLEAN CONSTRAINT SATISFACTION PROBLEMS 18
      2. PROCESS CONSTRAINT SATISFACTION PROBLEMS 19
      3. HEURISTIC SEARCH PROBLEMS 20
   E. CONCLUSIONS 33

III. A PROBLEM SOLVER FOR PROBLEMS STATED IN REF 37
   A. THE TRANSLATION PROBLEM 37
   B. THE ARF INTERPRETER 40
      1. THE USE OF VARIABLES AND CONSTRAINTS 40
      2. CASE ANALYSIS AND THE SUBPROBLEM SPACE 41
      3. SIZE OF THE SUBPROBLEM SPACE 42

IV. SOLVING CONSTRAINT SATISFACTION PROBLEMS 59
   A. DESIGNING A CONSTRAINT SATISIFIER 59
      1. VALUE ASSIGNMENT METHODS 59
      2. CONSTRAINT MANIPULATION METHODS 60
      3. COMBINING CONSTRAINT MANIPULATION AND VALUE
         ASSIGNMENT 60
      4. THE RESULTING DESIGN 61
   B. ARFVS CONSTRAINT SATISFACTION METHODS 78
      1. EXPRESSION EVALUATION AND SIMPLIFICATION 78
      2. ENTERING A VALUE INTO A CONTEXT 79
      3. CONSTRAINT PROCESSING 80
         3.1. PHASE 1 80
         3.2. PHASE 2 80
         3.3. PHASE 3 81
      4. EXAMPLE 81

V. SOLVING THE SEARCH PROBLEM IN THE SUBPROBLEM SPACE 97
   A. OPERATORS AND THEIR APPLICATION 97
      1. ARFVS OPERATOR FORMULATION 97
      2. OTHER OPERATOR FORMULATIONS 98
   B. METHODS FOR DIRECTING THE SEARCH 117
      1. ARFVS SEARCH STRATEGY 117
      2. RECOGNITION OF LOOPS 117
      3. USE OF AN INVERSE INTERPRETER 118
         3.1. CONDUCTING A TWO WAY SEARCH 118
         3.2. MEANS-ENDS ANALYSIS 118
         3.3. AN INITIALIZATION PROBLEM 119
TABLE OF FIGURES

II
1. CONVERSATION BETWEEN VSTUDENTV AND A HUMAN 5
2. (A) IS A TYPICAL PROBLEM FOR VSTUDENTV. (B) IS THE SET OF EQUATIONS DERIVED FROM (A). (C) IS ASSUMPTIONS MADE BY VSTUDENTV IN ORDER TO SOLVE (A). 6
3. PROBLEM SITUATION STATED IN THE PREDICATE CALCULUS 8
4. BACKUS NORMAL FORM DESCRIPTION OF REF 12
5A. ENGLISH STATEMENT OF THE MAGIC SQUARE PROBLEM 20
5B. REF STATEMENT OF THE MAGIC SQUARE PROBLEM 20
6A. ENGLISH STATEMENT OF A CRYPT-ADDITION PROBLEM 24
6B. REF STATEMENT OF A CRYPT-ADDITION PROBLEM 24
7. GENERAL FORM FOR REF STATEMENT OF A HEURISTIC SEARCH PROBLEM 26
8A. ENGLISH STATEMENT OF A WATER JUG PROBLEM 30
8B. REF STATEMENT OF A WATER JUG PROBLEM 30
9A. ENGLISH STATEMENT OF THE MONKEY PROBLEM 31
9B. REF STATEMENT OF THE MONKEY PROBLEM 31
10A. TEXTBOOK STATEMENT OF AN INTEGER LINEAR PROGRAMMING PROBLEM 35
10B. REF STATEMENT OF AN INTEGER LINEAR PROGRAMMING PROBLEM 35

III
1. CASE IN CRYPT-ADDITION PROBLEM WHERE CARRY VALUES ARE ALL ZERO 48
2A. THREE LEVELS OF THE STANDARD SEARCH TREE FOR THE MONKEY PROBLEM 55
2B. THREE LEVELS OF ARFVS SEARCH TREE FOR THE MONKEY PROBLEM 55
3A. ENGLISH STATEMENT OF THE MISSIONARIES AND CANNIBALS PROBLEM 58
3B. REF STATEMENT OF THE MISSIONARIES AND CANNIBALS PROBLEM 58

IV
1. ORDERING OF VARIABLES FOR GENERATION IN THE EXAMPLE CASE FROM THE CRYPT-ADDITION PROBLEM 65
2A. EXECUTIVE FOR CONSTRAINT SATISIFIER 76
2B. EXECUTIVE FOR CONSTRAINT SATISIFIER (CONTINUED) 77
3. ARFVS EXPRESSION EVALUATION EXECUTIVE 82
4A. EXAMPLE OF ARFVS EXPRESSION EVALUATION 84
4B. EXAMPLE OF ARFVS EXPRESSION EVALUATION 85

V
1. EXAMPLE OF ARFVS VFORV LOOP TRANSLATION 99
2. DIRECTED GRAPH FOR THE MONKEY PROBLEM 100
3. OPERATORS DEFINED IN THE DIRECTED GRAPH 102
4. ARFVS OPERATORS FOR THE MONKEY PROBLEM 105
5. OPERATORS FORMED BY SINGLE REF STATEMENTS 111
6. OPERATORS FROM THE CONSTRAINT SATISFACTION METHODS 115
7. ARFVS TOP EXECUTIVE 119
8A. ENGLISH STATEMENT OF A HYPOTHESIS FORMATION PROBLEM 122
8B. REF STATEMENT OF A HYPOTHESIS FORMATION PROBLEM 123
9. FIRST PORTION OF ARFVS SEARCH TREE FOR THE HYPOTHESIS FORMATION PROBLEM 125
10. SEARCH TREE PRODUCED BY ARF FOR THE MONKEY PROBLEM 128
11. TWO NONIDENTICAL EQUIVALENT CONTEXTS 131
12. INDICATION OF THE PORTION OF ARFVS SEARCH FOR THE SOLUTION TO THE MONKEY PROBLEM WHICH COULD HAVE BEEN AVOIDED BY THE USE OF THE PROPOSED LOOP RECOGNITION ALGORITHM 135
13. EXAMPLE TWO WAY SEARCH FOR THE MONKEY PROBLEM 143
14. COMBINATION OF A FORWARD AND A BACKWARD CONTEXT 144

VI

1. ARFVS VS ETV STATEMENT INTERPRETER 153
2A. ARFVS REPLACER SUBROUTINE 154
2B. ARFVS REPLACER SUBROUTINE (CONTINUED) 155
2C. ARFVS REPLACER SUBROUTINE (CONTINUED) 156
3. CASES PRODUCED BY THE MATCHING PROCESS 159

VII

1. REF STATEMENT OF A SORTING PROBLEM 165
2. REF STATEMENT OF A SORTING PROBLEM 167
3. CONTEXT AT END OF INTERPRETATION 169
4. CONTEXT AFTER ASSIGNMENT OF VALUE TO S(2) 170
5. CONTEXT AFTER ASSIGNMENT OF VALUE TO S(3) 171
6. CONTEXT AT END OF INTERPRETATION 174
7. CONTEXT AFTER ASSIGNMENT OF VALUE TO S(3) 175
8. SOLUTION CONTEXT 176
9. REF STATEMENT OF THE PICNIC PROBLEM 179
10. CONTEXT AT END OF INTERPRETATION 181
11. CONTEXT AFTER ASSIGNMENT OF VALUE TO S(5) 183
12. SOLUTION CONTEXT 185
13. ENGLISH STATEMENT OF THE CONFUSION OF PATENTS PROBLEM 188
14A. FIRST REF FORMULATION OF THE CONFUSION OF PATENTS PROBLEM 190
14B. SECOND REF FORMULATION OF THE CONFUSION OF PATENTS PROBLEM 191
15. CONTEXT AFTER INTERPRETATION THROUGH VFORV LOOP 195
<table>
<thead>
<tr>
<th></th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>16.</td>
<td>CONTEXT AT END OF INTERPRETATION</td>
<td>197</td>
</tr>
<tr>
<td>17.</td>
<td>SOLUTION CONTEXT</td>
<td>200</td>
</tr>
<tr>
<td>18.</td>
<td>REF STATEMENT OF THE 8 QUEENS PROBLEM</td>
<td>203</td>
</tr>
<tr>
<td>19.</td>
<td>BEGINNING OF ARF'S SEARCH FOR THE 8 QUEENS PROBLEM</td>
<td>205</td>
</tr>
<tr>
<td>20.</td>
<td>ARF'S BACKTRACK SEARCH FOR THE 8 QUEENS PROBLEM</td>
<td>208</td>
</tr>
<tr>
<td>21.</td>
<td>SEARCH TREE FOR THE CRYPT-ADDITION PROBLEM</td>
<td>211</td>
</tr>
<tr>
<td>22.</td>
<td>CONTEXT AT END OF INTERPRETATION</td>
<td>212</td>
</tr>
<tr>
<td>23.</td>
<td>SOLUTION CONTEXT</td>
<td>213</td>
</tr>
<tr>
<td>24.</td>
<td>ARF'S SEARCH FOR THE HYPOTHESIS FORMATION PROBLEM</td>
<td>220</td>
</tr>
<tr>
<td>25A.</td>
<td>FIRST REF FORMULATION OF THE THREE COINS PUZZLE</td>
<td>225</td>
</tr>
<tr>
<td>25B.</td>
<td>SECOND REF FORMULATION OF THE THREE COINS PUZZLE</td>
<td>226</td>
</tr>
<tr>
<td>25C.</td>
<td>THIRD REF FORMULATION OF THE THREE COINS PUZZLE</td>
<td>227</td>
</tr>
<tr>
<td>26.</td>
<td>CONTEXT AFTER THE THREE MOVES</td>
<td>231</td>
</tr>
<tr>
<td>27.</td>
<td>SOLUTION CONTEXT</td>
<td>232</td>
</tr>
<tr>
<td>28.</td>
<td>SEARCH TREE PRODUCED BY ARF FOR THE MONKEY PROBLEM</td>
<td>235</td>
</tr>
<tr>
<td>29.</td>
<td>REF STATEMENT OF THE TOWER OF HANOI PROBLEM</td>
<td>239</td>
</tr>
<tr>
<td>30.</td>
<td>CONTEXT AFTER FOUR BOAT TRIPS</td>
<td>244</td>
</tr>
</tbody>
</table>
I. INTRODUCTION

WE OFTEN WRITE COMPUTER PROGRAMS TO SOLVE PROBLEMS. FOR A PARTICULAR PROBLEM THE VALUE OF A PROBLEM SOLVING PROGRAM IS RELATED TO THE EASE WITH WHICH A PERSON CAN STATE THE PROBLEM TO THE PROGRAM AND THE EFFECTIVENESS OF THE PROGRAM'S PROBLEM SOLVING METHODS FOR FINDING A SOLUTION. IN GENERAL, THE VALUE OF A PROBLEM SOLVER CAN BE MEASURED IN TERMS OF THE RANGE OF PROBLEMS WHICH CAN BE EASILY STATED IN ITS INPUT LANGUAGE AND THE RANGE OF PROBLEMS FOR WHICH ITS PROBLEM SOLVING METHODS ARE EFFECTIVE. HENCE, TO BUILD A GOOD PROBLEM SOLVING PROGRAM WE MUST BE CONCERNED BOTH WITH THE GENERALITY OF ITS INPUT LANGUAGE AND THE EFFECTIVENESS AND GENERALITY OF ITS PROBLEM SOLVING METHODS. THIS PAPER IS A PROGRESS REPORT OF A CONTINUING RESEARCH EFFORT TO BUILD A PROBLEM SOLVER WITH THESE TWO CONCERNS IN MIND.

TO OBTAIN THE DESIRED GENERALITY AND EASE OF PROBLEM STATEMENT IN AN INPUT LANGUAGE WE PROPOSE EXTENDING A PROGRAMMING LANGUAGE TO FORM A NONDETERMINISTIC LANGUAGE WHICH IS SUITABLE FOR STATING PROBLEMS. THE EXTENSIONS PROVIDE FACILITIES FOR INDICATING THAT AN ELEMENT IS TO BE SELECTED FROM A SET AND FOR STATING CONSTRAINTS ON SUCH SELECTIONS WHILE ALLOWING PRESERVATION OF THE REPRESENTATIONAL POWER AND GENERALITY INHERENT IN THE BASE PROGRAMMING LANGUAGE. WE PRESENT A PARTICULAR INSTANCE OF SUCH AN
INPUT LANGUAGE, CALLED REF, AND EXPLORE THE RANGE OF PROBLEMS EXPRESSABLE IN IT. FINALLY, WE DISCUSS THE SCOPE AND LIMITATIONS OF NONDETERMINISTIC PROGRAMMING LANGUAGES AS INPUT LANGUAGES AND COMPARE THEM WITH THE INPUT LANGUAGES USED BY OTHER PROBLEM SOLVING PROGRAMS.

THE PROGRAM WE DESCRIBE, CALLED ARF, WAS DESIGNED TO ACCEPT PROBLEMS STATED IN REF AND TO DEAL EFFECTIVELY WITH THE DIVERSITY OF PROBLEMS EXPRESSIBLE IN THE LANGUAGE. ITS PROBLEM SOLVING POWER RESULTS FROM A COMBINATION OF CONSTRAINT SATISFACTION METHODS AND HEURISTIC SEARCH METHODS. WE PRESENT DISCUSSIONS OF ITS STRUCTURE, BEHAVIOR, AND EXTENSIBILITY WITH THE AIM OF GAINING INSIGHT INTO THE NATURE OF INTELLIGENT PROBLEM SOLVING ACTIVITY.

WE WILL PROCEED BY CONSIDERING THE INPUT LANGUAGE PROPOSAL IN CHAPTER II. CHAPTERS III THROUGH VI DESCRIBE AND DISCUSS THE PROBLEM SOLVING PROGRAM. CHAPTER VII PRESENTS A SET OF PROBLEMS AND DISCUSSES THEIR STATEMENT IN REF AND SOLUTION BY ARF. FINALLY, IN CHAPTER VIII WE SUMMARIZE THE RESULTS OF THE PROJECT AND CONSIDER THE DIRECTION IN WHICH IT IS MOVING.
II. INPUT LANGUAGE FOR A GENERAL PROBLEM SOLVER

A. THE PROBLEM OF DESIGNING AN INPUT LANGUAGE

The input language of a problem solving program has demands placed upon it both by the user of the program and by the designer of the program. The user must represent in the language the problems he has to be solved, and he wishes that problem statement task to be as easy as possible. The designer of the problem solver must devise programmable methods for solving problems stated in the language and he wishes those methods to be as effective as possible. These demands are often incompatible and one is forced to make tradeoffs between the two. To better see this incompatibility, consider two languages which are often suggested as desirable input languages for a general problem solver: English and the predicate calculus.

English (or any natural language) is a highly desirable input language for the user but an extremely difficult one for the program designer. The problem of enabling a program to understand a problem stated in English so that it can apply its problem solving methods still appears to be extremely difficult, but the desirability of such a problem solver with English as an input language continues to motivate research in
THAT DIRECTION. THE TYPICAL STRATEGY FOR THESE EFFORTS HAS BEEN TO
RESTRICT THE INPUT LANGUAGE TO SOME SUBSET OF ENGLISH OR TO
RESTRICT THE DOMAIN OF PROBLEMS WHICH THE PROGRAM WILL ATTEMPT TO
SOLVE. THE VSIRV QUESTION ANSWERING PROGRAM (RAFAEL, 1964) IS AN
EXAMPLE OF THE USE OF A SUBSET OF ENGLISH AS AN INPUT LANGUAGE.
IT RESTRICTS ENGLISH BY ALLOWING ONLY A GIVEN SET OF SENTENCE
FORMS, AND APPROACHES THE INTERPRETATION TASK BY USING A SEPARATE
INTERPRETATION ROUTINE FOR EACH SENTENCE FORM. A TYPICAL
INTERACTION BETWEEN VSIRV AND A HUMAN IS SHOWN IN FIGURE II.1. THE
VSIRV PROBLEM SOLVING PROGRAM (BOBROW, 1964) IS AN EXAMPLE OF
THE USE OF ENGLISH AS AN INPUT LANGUAGE FOR A RESTRICTED CLASS OF
PROBLEMS. VSIRV ATTEMPTS TO SOLVE STORY ALGEBRA PROBLEMS AS
FOUND IN HIGH-SCHOOL ALGEBRA TEXTBOOKS. THE RESTRICTED PROBLEM
DOMAIN MAKES POSSIBLE THE USE OF ALGORITHMIC TECHNIQUES FOR
PROBLEM INTERPRETATION AND SOLUTION. FIGURE II.2 ILLUSTRATES A
TYPICAL PROBLEM INTERPRETATION BY VSIRV.
MAN: EVERY KEYPUNCH-OPERATOR IS A GIRL
SIR: I UNDERSTAND
MAN: ANY GIRL IS AN EXAMPLE OF A PERSON
SIR: I UNDERSTAND
MAN: IS A KEYPUNCH-OPERATOR A PERSON?
SIR: YES
MAN: IS A PERSON A PERSON?
SIR: YES
MAN: IS A PERSON A GIRL?
SIR: SOMETIMES
MAN: IS A MONKEY A KEYPUNCH-OPERATOR?
SIR: INSUFFICIENT INFORMATION

FIGURE II.1. CONVERSATION BETWEEN SIR AND A HUMAN
(A)

The gas consumption of my car is 15 miles per gallon. The distance between Boston and New York is 250 miles. What is the number of gallons of gas used on a trip between New York and Boston?

(B)

1. \[(\text{Equal (distance between Boston and New York) (times 250 (miles))})\]
2. \[(\text{Equal x1 (number of gallons of gas used on trip between New York and Boston))}\]
3. \[(\text{Equal (gas consumption of my car) (quotient (times 15 (miles)) (times 1 (gallons))))}\]
4. \[(\text{Equal (distance) (times (gas consumption) (number of gallons of gas used)))}\]

(C)

1. Gas consumption = gas consumption of my car
2. Distance = distance between Boston and New York
3. Number of gallons of gas used = number of gallons of gas used on a trip between New York and Boston

Figure II.2. (A) is a typical problem for \text{\textvisiblespace}student\textvisiblespace. (B) is the set of equations derived from (A). (C) is assumptions made by \text{\textvisiblespace}student\textvisiblespace in order to solve (A).
THE PREDICATE CALCULUS IS A HIGHLY DESIRABLE INPUT LANGUAGE FOR THE DESIGNER OF A PROBLEM SOLVER BUT A DIFFICULT ONE FOR THE USER. ITS DESIRABILITY FOR THE DESIGNER CAN BE SEEN IN THE RICH HISTORY AND THE SUCCESS OF ATTEMPTS TO DEVELOP PROGRAMMABLE THEOREM PROVING TECHNIQUES. THE USER'S COMPLAINT IS THAT EVEN THOUGH THE PREDICATE CALCULUS IS IN SOME SENSE A "UNIVERSAL" LANGUAGE, IT IS OFTEN QUITE DIFFICULT AND CUMBERSOME TO USE FOR STATING PROBLEMS. THIS DIFFICULTY HAS BEEN DRAMATIZED IN ATTEMPTS BY MCCARTHY (1959), BLACK (1964), AND OTHERS TO USE THE PREDICATE CALCULUS TO REPRESENT PROBLEMS OF EVERYDAY REASONING. FIGURE II.3 SHOWS AN EXAMPLE OF SUCH A PROBLEM AND ITS STATEMENT IN THE PREDICATE CALCULUS.
CONSIDER A SITUATION IN WHICH A MONKEY IS IN A
ROOM WHERE A BUNCH OF BANANAS IS HANGING FROM
THE CEILING TOO HIGH TO REACH. IN THE CORNER
OF THE ROOM IS A BOX, AND THE SOLUTION TO THE
MONKEY'S PROBLEM IS TO MOVE THE BOX UNDER THE
BANANAS AND CLIMB ONTO THE BOX FROM WHICH THE
BANANAS CAN BE REACHED. THIS SITUATION IS
DESCRIBED BY MCCARTHY (1963) IN AN EXTENDED
FORM OF THE PREDICATE CALCULUS AS FOLLOWS:

1. \( \text{VU } \text{PLACE(U) CAN(MONKEY, MOVE(MONKEY, BOX, U))} \)
2. \( \text{VU VV VP MOVE(P, V, U) CAUSE(AT(V, U))} \)
3. \( \text{CAN(MONKEY, CLIMBS(MONKEY, BOX))} \)
4. \( \text{VU VV VP AT(V, U) CLIMBS(P, V) CAUSE(AT(V, U) \land}
    \text{ON(P, V))} \)
5. \( \text{PLACE(Under(BANANAS))} \)
6. \( \text{AT(BOX, Under(BANANAS)) \land ON(MONKEY, BOX) CAN(MONKEY,}
    \text{REACH(MONKEY, BANANAS))} \)
7. \( \text{VP VX REACH(P, X) CAUSE(HAS(P, X))} \)

PROPOSITION 1 STATES THAT IF U IS A PLACE THEN
THE MONKEY CAN MOVE THE BOX TO U. PROPOSITION
2 STATES THAT P MOVING V TO U CAUSES V TO BE
AT U. PROPOSITION 3 STATES THAT THE MONKEY CAN
CLIMB ON THE BOX. PROPOSITION 4 STATES THAT V
BEING AT U AND P CLIMBING ON V CAUSES V TO BE
AT U AND P TO BE ON V. PROPOSITION 5 STATES
THAT UNDER THE BANANAS IS A PLACE. PROPOSITION
6 STATES THAT IF THE BOX IS UNDER THE BANANAS
AND THE MONKEY IS ON THE BOX THEN THE MONKEY
CAN REACH THE BANANAS. PROPOSITION 7 STATES
THAT P REACHING X CAUSES P TO HAVE X.

FIGURE II.3. PROBLEM SITUATION STATED IN THE PREDICATE CALCULUS
B. AN INPUT LANGUAGE PROPOSAL

WE WISH TO PROPOSE HERE A FORM OF INPUT LANGUAGE WHICH HAS A
UNIVERSAL QUALITY, AS DOES THE PREDICATE CALCULUS, BUT WHICH
PROVIDES A MORE EQUITABLE COMPROMISE OF THE USERS AND DESIGNERS
DEMANDS THAN DOES EITHER ENGLISH OR THE PREDICATE CALCULUS. THE
ELEMENTS OF THE PROPOSED LANGUAGE ARE FORMALLY DEFINED SO THAT A
PROGRAM CAN ALGORITHMICALLY INTERPRET A PROBLEM STATEMENT, YET THE
REPRESENTATIONAL POWER OF THE LANGUAGE IS STRONG ENOUGH TO ALLOW
STATEMENT OF A DIVERSE CLASS OF PROBLEMS IN A NATURAL MANNER.

WE WILL CREATE A LANGUAGE FOR STATING PROBLEMS BY EXTENDING A
LANGUAGE DESIGNED FOR STATING SOLUTION PROCEDURES TO PROBLEMS
(I.E., A PROGRAMMING LANGUAGE). ONE STILL WRITES PROCEDURES IN THE
EXTENDED LANGUAGE, BUT THESE PROCEDURES MAY DEFINE PROBLEMS BY
INDICATING A SELECTION FROM A SPACE OF POTENTIAL SOLUTIONS AND
THEN A VERIFICATION THAT THE SELECTION SATISFIES THE REQUIREMENTS
OF THE PROBLEM FOR A SOLUTION. THE EXTENSIONS INVOLVE THE ADDITION
OF A VSELECTV FUNCTION AND A VCONDITIONV STATEMENT TO THE
PROGRAMMING LANGUAGE.

THE VSELECTV FUNCTION PROVIDES THE FACILITY FOR INDICATING
THAT A SELECTION IS TO BE MADE FROM A SPACE OF POTENTIAL
SOLUTIONS. IT REQUIRES ARGUMENTS WHICH DEFINE A SET AND THE VALUE
OF THE FUNCTION IS AN ELEMENT OF THAT SET. FOR EXAMPLE, IF THE
BASE PROGRAMMING LANGUAGE IS ^VALGOL^ THEN THE STATEMENT
^B:=SELECT (0, 9)^ COULD MEAN THAT B IS TO BE ASSIGNED AS A VALUE AN
INTEGER IN THE RANGE 0 TO 9.

THE ^CONDITION^ STATEMENT IS USED TO STATE A BOOLEAN
EXPRESSION WHICH MUST HAVE ^TRUE^ AS A VALUE. BY USING THE
^CONDITION^ STATEMENT ONE CAN VERIFY THAT A SELECTION IS A
SOLUTION TO THE PROBLEM BEING STATED. FOR EXAMPLE, CONSIDER THE
PROBLEM OF FINDING TWO INTEGERS IN THE RANGE 0 TO 9 WHOSE SUM IS
15. USING ^VALGOL^ AS THE BASE LANGUAGE AND THE INTEGER VALUED
^SELECT^ FUNCTION MENTIONED ABOVE, THE PROBLEM COULD BE STATED AS
FOLLOWS:

BEGIN;
INTEGER B, C;
B:=SELECT (0, 9);
C:=SELECT (0, 9);
CONDITION B+C=15;
END;

THE PROCEDURE (OR PROBLEM STATEMENT) SAYS, IN EFFECT, SELECT
INTEGER VALUES IN THE RANGE 0 TO 9 FOR EACH OF B AND C SUCH THAT
THEIR SUM IS 15.

WE WILL REFER TO SUCH A LANGUAGE AS A NONDETERMINISTIC
PROGRAMMING LANGUAGE. THIS TERMINOLOGY IS DERIVED FROM A PAPER ON
NONDETERMINISTIC ALGORITHMS BY FLOYD (1967) AND THE CONCEPT OF A
NONDETERMINISTIC AUTOMATON AS INTRODUCED BY RABIN AND SCOTT
THE USE OF THE TERM NONDETERMINISTIC IS NOT MEANT TO IMPUTE A PROBABILISTIC NATURE TO THE LANGUAGE, BUT RATHER THAT ONE MAY INCLUDE FREEDOM OF CHOICE OR MULTIPLE PATHS IN PROCEDURES WRITTEN IN THE LANGUAGE. ANY PROCESSOR WHICH EXECUTES A NONDETERMINISTIC PROCEDURE MUST MAKE CHOICES (OR SELECTIONS) SO THAT ALL "CONDITION" STATEMENTS ARE SATISFIED AND SO THAT THE PROCEDURE TERMINATES.

WE WILL EXPLORE THE REPRESENTATIONAL POWER AND SCOPE OF NONDETERMINISTIC INPUT LANGUAGES BY CONSIDERING THE PARTICULAR INPUT LANGUAGE CREATED FOR OUR PROBLEM SOLVING PROGRAM. THIS LANGUAGE, CALLED REF, USES AS ITS BASE A SIMPLE ALGOL-LIKE PROGRAMMING LANGUAGE WITH FACILITIES FOR SYMBOL AS WELL AS NUMERICAL MANIPULATION. WE WILL PROCEED BY DESCRIBING REF AND THEN EXPLORING ITS USAGE FOR STATING VARIOUS TYPES OF PROBLEMS.

C. THE REF INPUT LANGUAGE

FIGURE II.4 PROVIDES A BACKUS NORMAL FORM DESCRIPTION OF THE SYNTAX OF REF. THE FOLLOWING PARAGRAPHS DESCRIBE THE SEMANTICS OF REF.
PROCEDURE := BEGIN; STATEMENT STRING END;

STATEMENT STRING := STATEMENT | STATEMENT STRING STATEMENT

STATEMENT := LABEL | UNLabeled STATEMENT | UNlabeled STATEMENT

UNlabeled STATEMENT := GO TO LABEL, X | SET SLOT EXPRESSION TO EXPRESSION | SET VECTOR IDENTIFIER EXPRESSION TO EXPRESSION STRING | FOR IDENTIFIER -> INTEGER EXPRESSION DO TO LABEL; | IF BOOLEAN EXPRESSION THEN LABEL, X | CONDITION BOOLEAN EXPRESSION | GOTO INTEGER EXPRESSION

LABEL, X STRING := LABEL, X | LABEL, X STRING

EXPRESSION STRING := EXPRESSION | EXPRESSION STRING

EXPRESSION := SLOT EXPRESSION | IDENTIFIER EXPRESSION | INTEGER EXPRESSION | BOOLEAN EXPRESSION

SLOT EXPRESSION := v IDENTIFIER EXPRESSION | IDENTIFIER EXPRESSION INTEGER EXPRESSION | IDENTIFIER EXPRESSION IDENTIFIER EXPRESSION

IDENTIFIER EXPRESSION := SLOT EXPRESSION | IDENTIFIER | IDENTIFIER EXPRESSION

INTEGER EXPRESSION := IDENTIFIER EXPRESSION INTEGER EXPRESSION | SELECT INTEGER EXPRESSION | INTEGER EXPRESSION INTEGER EXPRESSION

BOOLEAN EXPRESSION := EXPRESSION | EXPRESSION EXPRESSION | EXPRESSION IDENTIFIER EXPRESSION | EXPRESSION INTEGER EXPRESSION | EXPRESSION BOOLEAN EXPRESSION | EXPRESSION BOOLEAN EXPRESSION | BOOLEAN EXPRESSION BOOLEAN EXPRESSION

LABEL, X := LABEL | END

LABEL := SYMBOL

(Continued on next page)

FIGURE II.4. BACKUS NORMAL FORM DESCRIPTION OF REF
<IDENTIFIER> := <SYMBOL>

<SYMBOL> := <LETTER><ALPHANUMERIC.STRING> | <LETTER>

<ALPHANUMERIC.STRING> := <ALPHANUMERIC> | <ALPHANUMERIC.STRING>
                          <ALPHANUMERIC>

<ALPHANUMERIC> := <LETTER> | <DIGIT> | .

<INTEGER> := <DIGIT> | <DIGIT.STRING><DIGIT>

<DIGIT> := 0 | 1 | ... | 9

<LETTER> := A | B | ... | Z

FIGURE II.4 (CONTINUED).
EACH IDENTIFIER IN A REF PROBLEM STATEMENT NAMES A 
DESCRIBABLE VECTOR; I.E., IF B IS AN IDENTIFIER THEN vB[3]v REFERS 
TO THE THIRD ELEMENT OF THE VECTOR B AND vCOLOR OF Bv REFERS TO 
THE VALUE OF THE ATTRIBUTE vCOLORv OF B. VECTOR ELEMENTS AND 
ATTRIBUTE VALUES MAY BE EITHER INTEGERS OR IDENTIFIERS. VECTOR 
eLEMENT NUMBERS MAY RANGE OVER THE NONNEGATIVE INTEGERS. THE 
CONVENTIONAL NOTION OF ASSOCIATING A VALUE WITH AN IDENTIFIER HAS 
NO MEANING IN REF SINCE EACH IDENTIFIER NAMES A DATA STRUCTURE 
(THE DESCRIBABLE VECTOR) RATHER THAN A SINGLE VALUE. TO PROVIDE A 
FACILITY FOR ASSOCIATING A VALUE WITH AN IDENTIFIER THE ZERO TH 
eLEMENT OF A VECTOR IS TREATED AS A SPECIAL CASE SYNTACTICALLY AND 
CAN BE DENOTED BY ENCLOSING THE IDENTIFIER IN ANGULAR BRACKETS, 
E.G. <B>.

PARENTHESIZED EXPRESSIONS MAY BE FORMED WITH THE OPERATORS 
AN OPERATOR HIERARCHY IS USED TO ORDER EXPRESSION EVALUATION, WITH 
SUBEXPRESSIONS HAVING EQUAL HIERARCHY BEING EVALUATED FROM RIGHT 
TO LEFT. THE FOLLOWING EXAMPLE ILLUSTRATES THE EVALUATION OF A REF 
EXPRESSION.
ASSUME THAT \( <A> \) IS B, \( <I> \) IS 1, \( B[1] \) IS C, \( <D> \) IS E, \( <E> \) IS F, F OF G IS H, AND C OF H IS 3. THEN EVALUATION OF THE EXPRESSION \( <A>[<I>] \) OF \( <<D>> \) OF G PROCEEDS AS FOLLOWS:

\[
\begin{align*}
&<A>[<I>] \text{ OF } <<D>> \text{ OF } G \\
&B[1] \text{ OF } <E> \text{ OF } G \\
&C \text{ OF } F \text{ OF } G \\
&C \text{ OF } H \\
&3
\end{align*}
\]

THE \texttt{vsetv} STATEMENT IS THE BASIC ASSIGNMENT STATEMENT IN REF.

THE SLCT EXPRESSION IN A \texttt{vsetv} STATEMENT (SEE FIGURE II.4) INDICATES THE VECTOR ELEMENT OR IDENTIFIER-ATTRIBUTE PAIR WHICH IS TO BE ASSIGNED A VALUE; E.g., TO SET THE ZEROth ELEMENT OF VECTOR B TO BE 3 ONE WRITES \texttt{vset <b> TO 3;} v. THE \texttt{vsetv} STATEMENT IS A MULTIPLE ASSIGNMENT STATEMENT WHICH SETS THE VALUES OF VECTOR ELEMENTS 1 THROUGH N OF THE IDENTIFIER INDICATED BY THE STATEMENT'S IDENTIFIER EXPRESSION; E.g., TO SPECIFY THAT M IS TO BE THE VECTOR \((2, 3, 5, 7, 11)\), ONE WOULD WRITE \texttt{vset.vector m TO 2,3,5,7,11;} v.

THE \texttt{vforv} STATEMENT IS USED TO DEFINE A LOOP IN A PROBLEM STATEMENT. THE LOOP CONSISTS OF ALL THE STATEMENTS FOLLOWING THE \texttt{vforv} STATEMENT UP TO AND INCLUDING THE STATEMENT HAVING THE LABEL INDICATED IN THE \texttt{vforv} STATEMENT. THE FIRST TIME THROUGH THE LOOP THE ZEROth VECTOR ELEMENT OF THE IDENTIFIER INDICATED IN THE \texttt{vforv}
STATEMENT IS ASSIGNED THE VALUE 1, AND EACH SUBSEQUENT TIME THROUGH THE LOOP THIS VALUE IS INCREMENTED BY ONE. THE VALUE OF THE INTEGER EXPRESSION IN THE \texttt{FOR} STATEMENT DEFINES THE NUMBER OF TIMES THE LOOP IS TO BE INTERPRETED.

THE \texttt{IF} STATEMENT IS USED TO INDICATE A CONDITIONAL TRANSFER. IF THE VALUE OF THE BOOLEAN EXPRESSION IN THE STATEMENT IS TRUE THEN THE INTERPRETER IS TO TRANSFER TO THE STATEMENT INDICATED BY THE LABEL; OTHERWISE, INTERPRETATION CONTINUES SEQUENTIALLY.

THE COMPUTED \texttt{GOTO} STATEMENT IS USED TO INDICATE AN N-WAY BRANCH. THE VALUE OF THE INTEGER EXPRESSION INDICATES THE STATEMENT TO WHICH CONTROL IS TO BE TRANSFERRED AS FOLLOWS: IF THE VALUE OF THE INTEGER EXPRESSION IS J THEN CONTROL IS TRANSFERRED TO THE STATEMENT WHOSE LABEL IS THE JTH MEMBER OF THE PARENTHESIZED LABEL LIST. FOR EXAMPLE, IF \texttt{<I>} IS 3 THEN THE STATEMENT \texttt{GOTO <I>} \texttt{(L1,L2,L3,L4)} WILL CAUSE INTERPRETATION TO CONTINUE AT STATEMENT L3.

\texttt{EXCL} IS A BOOLEAN VALUED FUNCTION WHICH IS USED TO INDICATE THAT A SET OF VALUES MUST BE DISTINCT; I.E. \texttt{EXCL(A1,A2,...,AN)} IS TRUE IF AND ONLY IF \texttt{AI\neq AJ} FOR ALL I AND J SUCH THAT \texttt{I\neq J}.

INTERPRETATION OF A REF PROCEDURE HALTS WHEN THE \texttt{END}
STATEMENT IS REACHED. THE \texttt{\textsc{end}} STATEMENT MAY BE LABELED AND
TRANSFERRED TO FROM A \texttt{\textsc{go to}} OR AN \texttt{\textsc{if}} STATEMENT.

IN A REF PROCEDURE ONE MAY INDICATE THAT THE PROBLEM SOLVER
IS TO SELECT AN INTEGER FROM SOME RANGE BY USING THE \texttt{\textsc{select}}
FUNCTION IN AN ASSIGNMENT STATEMENT. THE FUNCTION TAKES TWO INTE-
GER ARGUMENTS THAT DEFINE THE INTERVAL RANGE FROM WHICH AN INTEGER
IS TO BE SELECTED. FOR EXAMPLE, TO SPECIFY THAT THE PROBLEM SOLVER
IS TO ASSIGN AN INTEGER FROM THE RANGE $0,1,...,9$ TO $\langle \! \langle B \rangle \! \rangle$, ONE WOULD
WRITE \texttt{\textsc{set} $\langle B \rangle$ TO SELECT($0,9$)}.

THE \texttt{\textsc{condition}} STATEMENT IS USED TO REQUEST VERIFICATION OF
THE SELECTED SOLUTION. THE STATEMENT INDICATES TO THE INTERPRETER
THAT THE SELECTIONS IT HAS MADE MUST SATISFY THE STATEMENT\textsc{'}S
BOOLEAN EXPRESSION (I.E. GIVE IT THE VALUE \textsc{true}).

THIS COMPLETES THE DESCRIPTION OF THE REF LANGUAGE. THE
LANGUAGE IS A SIMPLE ONE LACKING REAL VALUED EXPRESSIONS, NUMERIC
OPERATORS SUCH AS MULTIPLICATION AND EXPONENTIATION, BLOCK
STRUCTURE, AND SUBROUTINES. THESE DEFICIENCIES MAKE REF UNSUITABLE
FOR REPRESENTING PROBLEM SITUATIONS WITH COMPLEX PROCESSES OR
COMPLEX OBJECTS, BUT THERE IS SUFFICIENT POWER IN THE LANGUAGE TO
ALLOW NATURAL REPRESENTATION OF AN INTERESTING RANGE OF PROBLEMS.
D. STATING PROBLEMS IN REF

IN THIS SECTION WE WILL EXPLORE THE CHARACTERISTICS OF REF PROBLEM STATEMENTS BY CONSIDERING THE REPRESENTATION OF THREE DIFFERENT FORMS OF PROBLEMS.

1. BOOLEAN CONSTRAINT SATISFACTION PROBLEMS

CONSIDER THE CLASS OF PROBLEMS WHICH HAVE THE FOLLOWING GENERAL FORM:

SELECT X1 FROM SET R1, X2 FROM SET R2, ..., AND XK FROM SET RK SUCH THAT THE CONSTRAINTS C1(X1,X2,...,XK), C2(X1,X2,...,XK), ..., CN(X1,X2,...,XK) ARE SATISFIED; WHERE C1,C2,...,CN ARE BOOLEAN EXPRESSIONS.

WE MIGHT CALL THIS THE CLASS OF BOOLEAN CONSTRAINT SATISFACTION PROBLEMS. MEMBERS OF THE CLASS INCLUDE PROBLEMS OF SOLVING SETS OF SIMULTANEOUS EQUATIONS, FINDING FEASIBLE SOLUTIONS TO RESOURCE ALLOCATION PROBLEMS, SORTING A LIST OF NUMBERS, AND VARIOUS PUZZLES.

THE REF \texttt{SELECT} FUNCTION AND \texttt{CONDITION} STATEMENT PROVIDE NATURAL MECHANISMS FOR STATING PROBLEMS FROM THIS CLASS. TO STATE ONE OF THESE PROBLEMS IN REF ONE MUST FIRST REPRESENT THE OBJECTS...
OR QUANTITIES OF THE PROBLEM AS REP DATA STRUCTURES. GIVEN THIS
REPRESENTATION, ONE WRITES A REP PROCEDURE CONSISTING OF A
SEQUENCE OF ASSIGNMENT STATEMENTS TO INDICATE THAT VALUES FOR THE
XI ARE TO BE SELECTED FROM THE SETS Ri FOLLOWED BY A SEQUENCE OF
\( \text{\texttt{\textbackslash vcondition\texttt{\textbackslash v}}} \) STATEMENTS TO INDICATE THE CONSTRAINTS WHICH THE
SELECTIONS MUST SATISFY.

A TYPICAL EXAMPLE OF A BOOLEAN CONSTRAINT SATISFACTION
PROBLEM IS THE MAGIC SQUARE PUZZLE STATED IN ENGLISH IN FIGURE
II.5A. THIS PROBLEM CAN BE STATED WITH RESPECT TO THE FORM GIVEN
ABOVE SUCH THAT K IS 9, EACH Ri IS THE SET OF INTEGERS 1, 2, \ldots, 9,
N IS 9, AND THE CONSTRAINTS CJ (X1, X2, \ldots, X9) ARE THE 8 SUMMATION
EQUATIONS AND THE REQUIREMENT THAT EACH INTEGER IN THE SOLUTION BE
DISTINCT. FIGURE II.5B SHOWS A STATEMENT OF THE PROBLEM IN REP
USING THIS FORMULATION. THE REP STATEMENT USES A VECTOR TO
REPRESENT THE MAGIC SQUARE MATRIX, A \texttt{\textbackslash vsetvector\texttt{\textbackslash v}} STATEMENT TO
SELECT THE SOLUTION, AND A SEQUENCE OF \texttt{\textbackslash vcondition\texttt{\textbackslash v}} STATEMENTS
CONTAINING THE CONSTRAINTS TO VERIFY THE SELECTIONS AS A SOLUTION.

REP STATEMENTS OF OTHER PROBLEMS FROM THIS CLASS CAN BE FOUND
IN CHAPTER VII. THESE INCLUDE THE PICNIC PROBLEM, A SORT PROBLEM,
AND THE 8 QUEENS PROBLEM.
ASSIGN ONE OF THE FIRST 9 POSITIVE INTEGERS TO EACH ELEMENT OF A 3-BY-3 MATRIX SUCH THAT NO INTEGER APPEARS MORE THAN ONCE IN THE MATRIX AND THE SUM OF THE INTEGERS IN EACH ROW, COLUMN, AND DIAGONAL IS 15.

FIGURE II.5A. ENGLISH STATEMENT OF THE MAGIC SQUARE PROBLEM

BEGIN;

SET VECTOR M TO SELECT(1, 9), SELECT(1, 9), SELECT(1, 9), SELECT(1, 9),
SELECT(1, 9), SELECT(1, 9), SELECT(1, 9), SELECT(1, 9), SELECT(1, 9);

CONDITION EXCL(M[1], M[2], M[3], M[4], M[5], M[6], M[7], M[8], M[9]);

END;

FIGURE II.5B. REP STATEMENT OF THE MAGIC SQUARE PROBLEM
2. PROCESS CONSTRAINT SATISFACTION PROBLEMS

Now consider a more general class of constraint satisfaction problems in which verification of the selected solution is described by a process rather than by a set of boolean expressions. Such problems have the following form:

select x₁ from set r₁, x₂ from set r₂, ..., and xₖ from set rₖ such that P(x₁, x₂, ..., xₖ) completes successfully; where P is a process requiring k parameters which has both a success and a failure exit.

We might call this the class of process constraint satisfaction problems. A broad range of problems can be described in this form. For example, the problem of teaching a perceptron to discriminate between two classes of objects can be described as a problem of selecting a set of parameters (i.e. the set of weights) for a process which applies each of the objects to the perceptron and tests to see if the correct discrimination is made.

To state such problems in ref one proceeds as with the boolean constraint satisfaction problems except that in this case the sequence of "condition" statements is replaced by the description of a process. Here we clearly see the advantages of
HAVING AVAILABLE IN AN INPUT LANGUAGE THE CONTROL STRUCTURE OF A
PROGRAMMING LANGUAGE.

A TYPICAL EXAMPLE OF A PROCESS CONSTRAINT SATISFACTION
PROBLEM IS THE CRYPT-ADDITION PROBLEM STATED IN ENGLISH IN FIGURE
II.6A. IN THIS PROBLEM THE VALUES SELECTED FOR THE LETTERS MUST
SATISFY THE CONDITIONS WHICH ARISE DURING THE PROCESS OF
COLUMN-BY-COLUMN ADDITION. NOTE THAT ALTHOUGH IT IS POSSIBLE TO
STATE THESE CONDITIONS IN THE FORM OF A SINGLE EQUATION, THE
TYPICAL STATEMENT OF CRYPT-ADDITION PROBLEMS STRONGLY SUGGESTS THE
COLUMN-BY-COLUMN ADDITION AND, IN FACT, MOST HUMANS WILL EXPLOIT
THE PROPERTIES OF THAT PROCESS WHEN SOLVING THE PROBLEMS.

WE PRESENT IN FIGURE II.6B A STATEMENT OF THE PROBLEM IN REF
IN WHICH THE VECTORS A1 AND A2 REPRESENT THE ADDENDS, THE VSUMV
VECTOR REPRESENTS THE SUM, AND THE L VECTOR LISTS EACH OF THE
LETTERS TO BE ASSIGNED VALUES IN THE PROBLEM. THE LOOP TERMINATING
AT L1 SELCTS VALUES FOR EACH OF THE LETTERS. THE VCONDITIONV
STATEMENTS FOLLOWING L1 VERIFY THAT EACH SELECTED VALUE IS UNIQUE
AND THAT THERE ARE NO LEADING ZEROS IN THE SOLUTION. THE LOOP
ENDING AT L2 REPRESENTS THE ADDITION PROCESS AND INDICATES THAT
THE SOLUTION MUST SATISFY THE CONDITIONS IMPOSED BY THAT PROCESS.
EACH TIME THROUGH THE LOOP A COLUMN IS ADDED, BEGINNING WITH THE
RIGHTMOST ONE. THE FINAL VCONDITIONV STATEMENT INDICATES THAT THE
VALUE OF $m$ MUST EQUAL THE CARRY INTO THE LEFTMOST COLUMN.

REF STATEMENTS OF OTHER PROBLEMS FROM THIS CLASS CAN BE FOUND IN CHAPTER VII. THESE INCLUDE A HYPOTHESIS FORMATION PROBLEM AND THE THREE COINS PROBLEM.
ASSIGN A DECIMAL DIGIT TO EACH OF THE LETTERS IN THE WORDS **SEND**, **MORE**, AND **MONEY** SUCH THAT WHEN THE LETTERS ARE REPLACED BY THE CORRESPONDING DIGITS THE FOLLOWING SUMMATION IS TRUE:

```
SEND
+MORE
-----
MONEY
```

No digit may be assigned to more than one letter, and leading zeros are not allowed in the numbers formed by **SEND**, **MORE**, and **MONEY**.

**FIGURE II.6A. ENGLISH STATEMENT OF A CRYPT-ADDITION PROBLEM**

```
BEGIN;
    SET VECTOR A1 TO X, S, E, N, D;
    SET VECTOR A2 TO X, M, O, R, E;
    SET VECTOR SUM TO M, O, N, E, Y;
    SET VECTOR L TO D, N, E, S, R, O, M, Y;
    FOR I = 1 TO 9 DO TO L1;
L1:  SET L[I] TO SELECT(0, 9);
    CONDITION ~(<M>=0) ^ (~<S>=0);
    SET <CARRY> TO 0;
    FOR J = 4 TO L2;
    SET <I> TO 6 - <I>;
    IF <A1[I]> + <A2[I]> + <CARRY> < 10 THEN L3;
    SET <CARRY> TO 1;
    GOTO L2;
        SET <CARRY> TO 0;
    GOTO L2;
L2:  CONDITION <M> = <CARRY>;
END;
```

**FIGURE II.6B. REF STATEMENT OF A CRYPT-ADDITION PROBLEM**
3. HEURISTIC SEARCH PROBLEMS

Now consider a third class of problems consisting of those which can be naturally stated in the heuristic search paradigm introduced by Newell and Ernst (1965). These problems may be stated by describing an initial object, a desired final object, and a set of operators for producing new objects from given objects. Newell and Ernst (1965) argue the generality of this paradigm and show that theorem proving, integration, sentence parsing, letter series completion, and various puzzles are members of the class.

One can state problems from this class in ref by writing a procedure which has the general form shown in the flow chart of Figure II.7. The procedure begins by creating a data structure which represents the initial object of the problem. Following this initialization phase is a loop which can be exited only when the desired final object has been realized. Each time through this loop an operator is selected, tested for applicability, and applied. Operator application is followed by a test for the final object which results in either a continuation in the loop or a transfer to \text{Vend.}
Figure II.7. General form for REF statement of a heuristic search problem
A distinguishing characteristic of the REF procedures which state these heuristic search problems is that their termination is not guaranteed. That is, in the procedures we considered earlier there were only a finite number of possible calls on the vselectv function and each possible combination of values for those calls led either to a vconditionv statement whose Boolean expression was false or to termination at vendv. In contrast, REF procedures which describe heuristic search problems may cause cycling to occur in a loop with new calls on the vselectv function being made during each cycle.

A typical example from this class of problems is the water jug problem (Mott-Smith, 1954) stated in English in Figure II.8A. For this problem an object can be described by an ordered pair of positive integers representing the number of gallons of water in each of the two jugs. Hence, (0,0) describes the initial object and any object whose description has a second element of 2 is a final object. There are six possible operators as follows:

1. Fill the first jug from the water source.
2. Fill the second jug from the water source.
3. Empty the first jug into the water sink.
4. Empty the second jug into the water sink.
5. Pour as much water as possible from the
SECOND JUG INTO THE FIRST JUG.

6. POUR AS MUCH WATER AS POSSIBLE FROM THE FIRST JUG INTO THE SECOND JUG.

ANY OF THE SIX OPERATORS ARE APPLICABLE TO ANY OBJECT.


FOR PROBLEMS OF THIS CLASS WHICH HAVE A LARGE NUMBER OF OPERATORS AND APPLICABILITY TESTS FOR EACH OPERATOR, ONE CAN OFTEN SHORTEN THE REF PROBLEM STATEMENT BY ADOPTING A MORE COMPLEX CONTROL STRUCTURE THAN THE ONE DISCUSSED ABOVE. FOR EXAMPLE, CONSIDER THE MONKEY PROBLEM (MCCARTHY, 1963) STATED IN ENGLISH IN FIGURE II.9A AND STATED IN REF IN FIGURE II.9E. WE HAVE MODELED THE SITUATION IN THE REF PROCEDURE BY ASSUMING THREE LOCATIONS ON THE FLOOR: \( X_1 \), \( X_2 \), AND \( \text{UNDER.BANANAV} \). THESE LOCATIONS ARE SUFFICIENT TO EXPRESS THE GENERAL SITUATION WHERE THE MONKEY IS NOT AT THE BOX AND NEITHER THE MONKEY NOR THE BOX IS UNDER THE BANANAS. WE ALSO ASSUME TWO VERTICAL LOCATIONS: \( \text{ON.FLOORV} \) AND \( \text{ON.BOXV} \). ONLY THE MONKEY CAN CHANGE VERTICAL POSITIONS AND THESE

1. - 3. THE MONKEY WALKS TO A LOCATION ON THE FLOOR.

4. - 6. THE MONKEY MOVES THE BOX TO A LOCATION ON THE FLOOR.

7. THE MONKEY CLIMBS ON THE BOX.

8. THE MONKEY STEPS DOWN FROM THE BOX.

9. THE MONKEY GETS THE BANANAS.

THE DESIRED FINAL STATE IS THE ONE WHICH RESULTS FROM APPLICATION OF THE NINTH OPERATOR, GET BANANAS.
GIVEN A FIVE GALLON JUG AND AN EIGHT GALLON JUG, HOW CAN PRECISELY TWO GALLONS BE PUT INTO THE FIVE GALLON JUG. SINCE THERE IS A SINK NEARBY, A JUG CAN BE FILLED FROM THE TAP AND CAN BE EMPTIED BY POURING ITS CONTENTS DOWN THE DRAIN. WATER CAN BE POURED FROM ONE JUG INTO ANOTHER, BUT NO MEASURING DEVICES ARE AVAILABLE OTHER THAN THE JUGS THEMSELVES.

FIGURE II.8A. ENGLISH STATEMENT OF A WATER JUG PROBLEM

BEGIN;
   SET <A> TO 0;
   SET <B> TO 0;
   L10: SET <J> TO SELECT (1,6);
          GOTO <J> (L1,L2,L3,L4,L5,L6);
   L1: SET <A> TO 8;
          GOTO L7;
   L2: SET <B> TO 5;
          GOTO L7;
   L3: SET <A> TO 0;
          GOTO L7;
   L4: SET <B> TO 0;
          GOTO L7;
   L5: IF 8 < <A> + <B> THEN L9;
          SET <A> TO <A> + <B>;
          GOTO L4;
   L9: SET <B> TO <A> + <B> - 8;
          GOTO L1;
   L6: IF 5 < <A> + <B> THEN L8;
          SET <B> TO <A> + <B>;
          GOTO L3;
   L8: SET <A> TO <A> + <B> - 5;
          GOTO L2;
   L7: IF ~(<B> = 2) THEN L10;
END;

FIGURE II.8B. REF STATEMENT OF A WATER JUG PROBLEM
IN A ROOM IS A MONKEY, A BOX, AND SOME BANANAS HANGING FROM THE CEILING. THE MONKEY WANTS TO EAT THE BANANAS, BUT HE CANNOT REACH THEM UNLESS HE IS STANDING ON THE BOX WHEN IT IS SITTING UNDER THE BANANAS. HOW CAN THE MONKEY GET THE BANANAS.

FIGURE II.9A. ENGLISH STATEMENT OF THE MONKEY PROBLEM

BEGIN;
   SET VECTOR X TO X1, X2, UNDER. BANANAS;
   SET VECTOR Y TO ON. FLOOR, ON. BOX;
   SET VECTOR MONKEY TO X1, ON. FLOOR;
   SET VECTOR BOX TO X2, ON. FLOOR;
   WALK: SET MONKEY[1] TO X[SELECT(1, 3)];
   IF ~(MONKEY[1] = BOX[1]) THEN WALK;
   L1: SET <M> TO SELECT(1, 3);
   GOTO <M> (WALK, CLIMB, MOVE BOX);
   CLIMB: SET MONKEY[2] TO ON. BOX;
   IF ~(MONKEY[1] = UNDER. BANANAS) THEN STEP. DOWN;
   SET <M> TO SELECT(1, 2);
   GOTO <M> (GET BANANAS, STEP. DOWN);
   STEP. DOWN: SET MONKEY[2] TO ON. FLOOR;
   GOTO L1;
   MOVE BOX: SET MONKEY[1] TO X[SELECT(1, 3)];
   SET BOX[1] TO MONKEY[1];
   GOTO L1;
   GET BANANAS: END;

FIGURE II.9B. REF STATEMENT OF THE MONKEY PROBLEM
This problem could be stated in REP using the form discussed above and indicated in Figure II.7. Such a REP statement would have a loop containing a nine-way branch and would have applicability tests preceding the application of each operator. For example, the applicability test for the get bananas operator would be the following two "condition" statements:

```
CONDITION MONKEY[2]=ON.BOX;
CONDITION MONKEY[1]=UNDER.BANANAS;
```

Alternatively, one can construct the REP procedure so that at each step only those operators which are applicable can be selected. This is the strategy adopted in writing the procedure of Figure II.9b. For example, if the monkey climbs on the box and is not under the bananas then he must step down. If he climbs on the box and is under the bananas, then he has the choice of getting the bananas or stepping down. This situation is represented in the REP procedure in lines 10-13. Another way in which the number of statements in this REP procedure has been reduced is by grouping the "walk" operators together and the "move.fox" operators together. For example, the selection of a walk operator is a two stage process; the monkey first chooses to walk and then chooses where to walk.

Chapter VII contains other problems from this class which we have stated in REP. These include the missionaries and cannibals
PROBLEM AND THE TOWER OF HANOI PROBLEM.

E. CONCLUSIONS

BY LOOKING AT A PARTICULAR INPUT LANGUAGE (I. E., REP) WE HAVE SEEN THE REPRESENTATIONAL POWER OF NONDETERMINISTIC LANGUAGES FORMED AS EXTENSIONS TO PROGRAMMING LANGUAGES. THE SUITABILITY OF A NONDETERMINISTIC LANGUAGE FOR A GIVEN CLASS OF PROBLEMS DEPENDS ON THE CHARACTERISTICS OF THE BASE PROGRAMMING LANGUAGE. THAT IS, THE USER MUST REPRESENT THE OBJECTS, PROCESSES, AND CONSTRAINTS OF A PROBLEM IN TERMS OF THE DATA STRUCTURES, STATEMENTS, EXPRESSIONS, AND CONTROL STRUCTURES AVAILABLE IN THE BASE LANGUAGE. THE EASE WITH WHICH THIS CAN BE DONE WILL DEPEND ON HOW WELL ORIENTED THE PROGRAMMING LANGUAGE IS TOWARD THE PROBLEM. THESE ARE THE SAME CONSIDERATIONS INVOLVED IN DETERMINING THE SUITABILITY OF A PROGRAMMING LANGUAGE FOR STATING AN ALGORITHM. HENCE, ONE MAY TAILOR AN INPUT LANGUAGE TOWARD PROBLEM CLASSES OF PARTICULAR INTEREST JUST AS IS DONE IN THE DESIGN OF PROGRAMMING LANGUAGES.

THE SUITABILITY OF AN INPUT LANGUAGE ALSO DEPENDS ON THE FORM OF ITS \texttt{SELECT} FUNCTIONS. ONE MIGHT WANT A DIFFERENT \texttt{SELECT} FUNCTION FOR EACH DATA TYPE IN THE LANGUAGE TO FACILITATE SELECTION OF INTEGERS, REALS, VECTORS, ARRAYS, LISTS, SYMBOLS,
SUBROUTINES, ETC. ONE MIGHT ALSO WISH TO ALLOW SELECTION FROM AN INFINITE RANGE SUCH AS THE POSITIVE INTEGERS, OR TO ALLOW THE PARAMETERS OF A SELECT FUNCTION TO BE EXPRESSIONS, OR TO HAVE A \texttt{\textbackslash select} FUNCTION WHICH SELECTS FROM A LIST OF INDEFINITE LENGTH, OR TO HAVE AN EXCLUSIVE \texttt{\textbackslash select} FUNCTION WHICH RETURNS A DIFFERENT VALUE EACH TIME IT IS CALLED.

ONE MIGHT ALSO CONSIDER OTHER EXTENSIONS BESIDES THE \texttt{\textbackslash select} FUNCTION AND THE \texttt{\textbackslash condition} STATEMENT. AN INTERESTING EXAMPLE WOULD BE A \texttt{\textbackslash maximize} OR A \texttt{\textbackslash minimize} STATEMENT. AN EXPRESSION WHOSE VALUE WAS TO BE OPTIMIZED WOULD BE INCLUDED IN THE STATEMENT AND IT WOULD BE A NUMERICAL VALUED EXPRESSION CONTAINING QUANTITIES WHICH WERE PREVIOUSLY SET BY CALLS ON THE \texttt{\textbackslash select} FUNCTION. SUCH A STATEMENT WOULD CAUSE A PROBLEM SOLVER TO CONSIDER THE ENTIRE SET OF SELECTIONS WHICH SATISFY THE OTHER CONSTRAINTS OF A PROBLEM AND DETERMINE ONE WHICH OPTIMIZES THE EXPRESSION OF THE \texttt{\textbackslash maximize} OR \texttt{\textbackslash minimize} STATEMENT. THE ADDITION OF SUCH A \texttt{\textbackslash maximize} STATEMENT AND A MULTIPLICATION OPERATOR TO \texttt{ref} WOULD ALLOW THE NATURAL STATEMENT OF INTEGER PROGRAMMING PROBLEMS. FOR EXAMPLE, FIGURE II.10A SHOWS AN INTEGER PROGRAMMING PROBLEM AS STATED IN A TEXTBOOK (HADLEY, 1964), AND FIGURE II.10B SHOWS THE SAME PROBLEM STATED IN THE EXTENDED \texttt{ref}.
Solve the following problem:

\[
\begin{align*}
3x_1 + 2x_2 & \leq 10, \\
2x_1 + 4x_2 & \leq 10, \\
x_1, x_2 & \geq 0, \quad x_1, x_2 \text{ integers,} \\
\text{max } Z = 3x_1 + 4x_2.
\end{align*}
\]

**Figure II.10A.** Textbook statement of an integer linear programming problem

begin;
set \langle A1 \rangle \text{ to select(0,10)};
set \langle A2 \rangle \text{ to select(0,10)};
condition \langle A1 \rangle + 2\langle A2 \rangle < 10;
condition \langle A1 \rangle + \langle A2 \rangle < 11;
maximize \langle A1 \rangle + 4\langle A2 \rangle;
end;

**Figure II.10B.** PFP statement of an integer linear programming problem
THESE SUGGESTIONS SERVE TO ILLUSTRATE THE RANGE OF POSSIBILITIES AVAILABLE FOR DESIGNING NONDETERMINISTIC PROGRAMMING LANGUAGES. GIVEN SUCH LANGUAGES WE MUST CONSIDER WHETHER WE CAN WRITE EFFECTIVE PROGRAMS FOR SOLVING PROBLEMS STATED IN THEM. WE HAVE WRITTEN SUCH A PROGRAM FOR PROBLEMS STATED IN REF AND WE WILL BEGIN THE DESCRIPTION OF THAT PROGRAM IN THE NEXT CHAPTER. IN THE FINAL CHAPTER OF THE PAPER WE WILL CONSIDER THE PROBLEM SOLVING DIFFICULTIES INTRODUCED BY THE MORE GENERAL INPUT LANGUAGES SUGGESTED IN THIS SECTION.
III. A PROBLEM SOLVER FOR PROBLEMS STATED IN REF

A. THE TRANSLATION PROBLEM

In our discussion of the requirements for designing a problem statement language for a general problem solver we observed that the language must allow effective problem solving methods to be programmed for problems stated in the language. Any given method requires that a problem be stated in a particular form before it can be applied to the problem. For example, consider the method which consists of generating each possible solution to a problem and testing whether the generated element satisfies the requirements for a solution. If a problem solver is to apply this method it must translate a problem into a generatable set of possible solutions and a solution test applicable to the generated solution candidates (see Newell, 1968, for a further discussion of the requirements made by general problem solving methods).

Hence, the first task in the design of a problem solver which uses REF as an input language is to devise ways of extracting the necessary information from a REF problem statement to translate the problem into a form which will enable the application of effective problem solving methods. This is the task faced by all problem solvers which do not use an input language tailored to
PARTICULAR PROBLEM SOLVING METHODS. FOR EXAMPLE, THE STUDENT PROGRAM DISCUSSED EARLIER MUST TRANSLATE ITS PROBLEMS FROM ENGLISH INTO A SET OF SIMULTANEOUS EQUATIONS, THE REQUIRED FORM FOR ITS SOLVING METHOD.

OUR PROBLEM SOLVER MUST BE ABLE TO EXTRACT THE MEANING OF A REPROCEDURE AND INTERPRET THIS MEANING TO PROBLEM SOLVING METHODS. THE PROBLEM OF DESIGNING AUTOMATIC MEANS FOR EXTRACTING MEANING FROM A PROCEDURE WRITTEN IN A PROGRAMMING LANGUAGE HAS BEEN STUDIED BY M CCARTHY (1962), FLOYD (1966), AND OTHERS. MOST OF THESE EFFORTS HAVE BEEN DIRECTED TOWARD THE DEVELOPMENT OF AUTOMATIC TECHNIQUES FOR PROVING THAT A PROGRAM DOES WHAT THE WRITER CLAIMS IT DOES. FLOYD'S WORK WITH NONDETERMINISTIC ALGORITHMS (1967), HOWEVER, IS DIRECTLY RELATED TO THE TASK OF TRANSLATING REP PROBLEM STATEMENTS INTO A SOLVABLE FORM. HE SHOWS THAT A PROCEDURE WRITTEN IN A NONDETERMINISTIC PROGRAMMING LANGUAGE CAN BE AUTOMATICALLY TRANSLATED INTO A PROCEDURE IN THE BASE PROGRAMMING LANGUAGE WHICH FINDS ACCEPTABLE VALUES FOR THE SELECT FUNCTION CALLS BY USING A BACKTRACKING ALGORITHM (GOLOMB, 1965).

THE BACKTRACKING ALGORITHM CONDUCTS A DEPTH-FIRST SEARCH AS FOLLOWS. DURING NORMAL EXECUTION A CALL ON THE SELECT FUNCTION PRODUCES AS A VALUE THE FIRST INTEGER IN THE RANGE SPECIFIED BY
THE PARAMETERS TO THE CALL (E.G., THE CALL \texttt{\textbackslash vselect(0,9)}\texttt{\textbackslash v} WOULD PRODUCE THE VALUE 0). EXECUTION CONTINUES UNTIL A \texttt{\textbackslash vcondition\textbackslash v} STATEMENT IS ENCOUNTERED WHOSE BOOLEAN EXPRESSION IS FALSE OR UNTIL \texttt{\textbackslash vend\textbackslash v} IS REACHED. IN THE CASE WHERE \texttt{\textbackslash vend\textbackslash v} IS REACHED, THE VALUES ASSIGNED TO THE \texttt{\textbackslash vselect\textbackslash v} FUNCTION CALLS CONSTITUTE A SOLUTION. IN THE CASE WHERE A \texttt{\textbackslash vcondition\textbackslash v} STATEMENT'S BOOLEAN EXPRESSION IS FALSE, THE ALGORITHM BACKTRACKS TO THE LAST CALL OF THE \texttt{\textbackslash vselect\textbackslash v} FUNCTION AND ATTEMPTS TO INCREMENT THE VALUE OF THAT CALL BY ONE. IF THE NEW VALUE IS IN THE RANGE SPECIFIED FOR THE CALL THEN EXECUTION AGAIN CONTINUES NORMALLY. IF THE NEW VALUE DOES NOT BELONG TO THE RANGE OF THE CALL, THEN A SECOND BACKTRACKING STEP OCCURS TO THE NEXT-TO-LAST \texttt{\textbackslash vselect\textbackslash v} FUNCTION CALL. A SIMILAR ATTEMPT IS MADE TO INCREMENT THE VALUE OF THAT CALL. THE ALGORITHM CONTINUES IN THIS MANNER UNTIL A SOLUTION HAS BEEN FOUND OR UNTIL THE VALUE OF THE FIRST \texttt{\textbackslash vselect\textbackslash v} FUNCTION CALL CANNOT BE FURTHER INCREMENTED.

IN THIS CASE THE PROBLEM SOLVING METHOD IS THE SYSTEM WHICH EXECUTES THE TRANSLATED ALGORITHM. FOR THIS METHOD TO SOLVE A PROBLEM IT REQUIRES AS INPUT AN ALGORITHM WRITTEN IN A GIVEN PROGRAMMING LANGUAGE WHICH WHEN EXECUTED WILL PRODUCE A SOLUTION TO THE PROBLEM. THE TRANSLATION PROCESS DEFINED BY FLOYD CREATES THE REQUIRED PROGRAM TO SOLVE THE PROBLEM JUST AS A HUMAN PROGRAMMER WOULD.
THE QUESTION ARISES AS TO WHETHER WE CAN PROGRAM OTHER
TRANSLATION SCHEMES FOR REF PROBLEM STATEMENTS TO ALLOW THE
APPLICATION OF METHODS MORE POWERFUL THAN ONE WHICH EXECUTES A
BACKTRACKING ALGORITHM. FOR EXAMPLE, IT WOULD BE DESIRABLE TO HAVE
A TRANSLATION OF THE REF STATEMENT OF THE MAGIC SQUARE PROBLEM
SHOWN IN FIGURE II.5B SUCH THAT TECHNIQUES FOR SOLVING
SIMULTANEOUS EQUATIONS COULD BE APPLIED TO THE EQUALITY
CONSTRAINTS, OR A TRANSLATION OF THE REF STATEMENT OF THE MONKEY
PROBLEM SHOWN IN FIGURE II.9B SUCH THAT HEURISTIC SEARCH
TECHNIQUES COULD BE APPLIED TO FIND THE DESIRED SEQUENCE OF
ACTIONS FOR THE MONKEY. THE PROBLEM SOLVER WE HAVE WRITTEN, CALLED
ARP, CONTAINS SUCH TRANSLATION SCHEMES. IT IS ABLE TO TRANSLATE
REF PROBLEM STATEMENTS SO THAT CONSTRAINT SATISFACTION METHODS AND
HEURISTIC SEARCH METHODS CAN BE EFFECTIVELY APPLIED. IN THE
REMAINDER OF THIS CHAPTER WE WILL DESCRIBE THE TRANSLATION
MECHANISMS OF ARP AND CHARACTERIZE THE FORMS IN WHICH THE PROBLEMS
ARE PRESENTED TO THE PROBLEM SOLVING METHODS. IN CHAPTERS IV, V,
AND VI WE WILL DISCUSS THE METHODS THEMSELVES.

B. THE ARP INTERPRETER

1. THE USE OF VARIABLES AND CONSTRAINTS

ARP TRANSLATES A PROBLEM STATED IN REF BY USING AN
INTERPRETER WHICH ACTS MUCH LIKE A STANDARD PROGRAMMING LANGUAGE INTERPRETER. THE CONTEXT IN WHICH THE INTERPRETER IS OPERATING AT ANY GIVEN TIME IS REPRESENTED BY A SINGLE DATA STRUCTURE. THIS CONTEXT STRUCTURE CONTAINS A POINTER TO THE NEXT STATEMENT TO BE INTERPRETED AND CONTAINS THE VECTOR AND THE ATTRIBUTE-VALUE PAIRS OF EACH IDENTIFIER WHICH HAS BEEN ENCOUNTERED DURING THE INTERPRETATION.

THE ARF INTERPRETER DIFFERS FROM A STANDARD INTERPRETER IN THE WAY IT DEALS WITH VSELECTV FUNCTION CALLS AND VCONDITIONV STATEMENTS. WHEN IT ENCOUNTERS AN ASSIGNMENT STATEMENT CONTAINING A VSELECTV FUNCTION CALL, IT DOES NOT ASSIGN AN INTEGER VALUE TO THE CALL. INSTEAD, IT CREATES A VARIABLE TO REPRESENT THE SELECTION AND TREATS THE NAME OF THAT VARIABLE AS THE VALUE OF THE CALL. THE NAME GIVEN TO THE VARIABLE CREATED BY THE ITH CALL OF THE VSELECTV FUNCTION IS S(I). WHEN A VARIABLE IS DEFINED BY THE INTERPRETER IT IS ENTERED INTO THE CONTEXT ALONG WITH A LIST CONTAINING EACH OF THE VALUES IN ITS RANGE.

THE USE OF VARIABLES IN THIS WAY IMPLIES THAT WHEN THE INTERPRETER EVALUATES A REP EXPRESSION, THE VALUE MAY BE ANOTHER EXPRESSION CONTAINING VARIABLES RATHER THAN AN INTEGER, IDENTIFIER, VTRUEV, OR VFALSEV. THE INTERPRETER MAY STORE THESE EXPRESSIONS IN THE CONTEXT AS VECTOR ELEMENTS OR AS VALUES OF
Attributes. For example, after interpretation of the `set vector` statement in the magic square problem each element of the vector \( m \) is a variable name rather than an integer, and the interpreter's context structure has the following form:

**Context**

**Data Structure**

`m`  
Vector: \( S(1), S(2), S(3), S(4), S(5), S(6), S(7), S(8), S(9) \)

**Variables**

\( S(1) \)  
Range: \( 1, 2, 3, 4, 5, 6, 7, 8, 9 \)

\( S(2) \)  
Range: \( 1, 2, 3, 4, 5, 6, 7, 8, 9 \)

\( \ldots \)

\( S(9) \)  
Range: \( 1, 2, 3, 4, 5, 6, 7, 8, 9 \)

When the interpreter encounters a `condition` statement it evaluates the boolean expression. If the evaluation produces `true`, then no other action is taken. If the evaluation produces `false`, then the interpreter indicates that no solution exists and halts. If the evaluation produces an expression containing variable names, then that expression is entered into the context as a constraint on the values of those variables. Hence, after interpreting the `condition` statements of the magic square problem, the context has the following form:
CONTEXT

DATA STRUCTURE M

VECTOR: S(1), S(2), S(3), S(4), S(5), S(6), S(7), S(8), S(9)

VARIABLES
S(1)
   RANGE: 1, 2, 3, 4, 5, 6, 7, 8, 9
S(2)
   RANGE: 1, 2, 3, 4, 5, 6, 7, 8, 9
   .
   .
S(9)
   RANGE: 1, 2, 3, 4, 5, 6, 7, 8, 9

CONSTRAINTS
S(1)+S(2)+S(3)=15
S(4)+S(5)+S(6)=15
   .
   .
S(3)+S(5)+S(7)=15
EXCL(S(1), S(2), S(3), S(4), S(5), S(6), S(7), S(8), S(9))

WHEN THE INTERPRETER REACHES VENDV, THE CONTEXT STRUCTURE
REPRESENTS THE COMPLETED TRANSLATION OF A PROBLEM. THE PROBLEM IS
IN THE FORM OF A BOOLEAN CONSTRAINT SATISFACTION PROBLEM IN THAT
THERE ARE A SET OF VARIABLES TO BE ASSIGNED VALUES, A RANGE OF
POSSIBLE VALUES FOR EACH VARIABLE, AND A SET OF CONSTRAINTS WHICH
THE VARIABLE VALUES MUST SATISFY. WITH THE PROBLEM STATED IN THIS
FORM, PROBLEM SOLVING METHODS CAN BE APPLIED WHICH ATTEMPT TO
DEDUCE VALUES FOR THE VARIABLES BY MANIPULATING THE CONSTRAINTS.
ARF CONTAINS A SET OF SUCH METHODS FOR SOLVING CONSTRAINT
SATISFACTION PROBLEMS WHICH ACCEPT AS INPUT A PROBLEM STATED IN
THE FORM OF A CONTEXT STRUCTURE PRODUCED BY THE INTERPRETER. WE
WILL DISCUSS IN THE NEXT CHAPTER HOW THESE METHODS ARE APPLIED TO
FIND ACCEPTABLE VALUES FOR A CONTEXT'S VARIABLES.

2. CASE ANALYSIS AND THE SUBPROBLEM SPACE

THE ARF INTERPRETER IS NOT ALWAYS ABLE TO TRANSLATE A PROBLEM
STATED IN REF INTO A SINGLE CONSTRAINT SATISFACTION PROBLEM
REPRESENTED AS A CONTEXT STRUCTURE. THE DIFFICULTY CAN OCCUR
DURING THE INTERPRETATION OF AN IF STATEMENT, A COMPUTED GOTO STATEMENT, OR AN ASSIGNMENT STATEMENT AS DESCRIBED BELOW.

IN THE INTERPRETATION OF AN IF STATEMENT, EVALUATION OF THE
BOOLEAN EXPRESSION IN THE STATEMENT MAY PRODUCE ANOTHER EXPRESSION
RATHER THAN TRUE OR FALSE. WHEN THIS SITUATION ARISES, PROVISION MUST BE MADE FOR CONSIDERING BOTH THE CASE WHERE THE
BRANCH IS TAKEN AND THE CASE WHERE IT IS NOT. THE INTERPRETER CAN
PERFORM THE REQUIRED CASE ANALYSIS BY SAVING A COPY OF THE CURRENT
CONTEXT AND EITHER ADDING THE BOOLEAN EXPRESSION TO THE CURRENT
CONTEXT AS A CONSTRAINT AND TAKING THE BRANCH OR ADDING THE
NEGATION OF THE BOOLEAN EXPRESSION TO THE CURRENT CONTEXT AS A
CONSTRAINT AND IGNORING THE BRANCH. THE SAVED CONTEXT COPY CAN BE
USED AT SOME LATER POINT IN THE PROCESSING FOR CONSIDERATION OF
THE SECOND CASE.

DURING THE INTERPRETATION OF A COMPUTED \texttt{GOTO} STATEMENT EVALUATION OF THE INTEGER EXPRESSION IN THE STATEMENT MAY PRODUCE ANOTHER EXPRESSION RATHER THAN AN INTEGER. IN THIS SITUATION EACH POSSIBLE BRANCH MUST BE CONSIDERED. AS WITH THE \texttt{IF} STATEMENT THE INTERPRETER CAN SAVE A COPY OF THE CURRENT CONTEXT AS IT CONSIDERS A BRANCH AND THEN RETURN TO CONSIDER ANOTHER OF THE BRANCHES LATER IN THE PROCESSING. WHEN THE INTERPRETER CONSIDERS ONE OF THE BRANCHES AT A COMPUTED \texttt{GOTO} STATEMENT, IT ADDS TO THE CONTEXT A CONSTRAINT EQUATING THE STATEMENT'S INTEGER EXPRESSION TO THE INTEGER VALUE WHICH IT MUST HAVE FOR THE BRANCH TO BE TAKEN.

THE \texttt{\textbackslash vsetv} STATEMENT WAS INTERPRETED (I.E., 2). DURING THE \texttt{\textbackslash vsetv}
STATEMENT PROCESSING THE INTERPRETER MUST MATCH A(S(3)) WITH A[1]
AND CREATE TWO CASES. FOR CASE 1 THE CONSTRAINT VS(3) \textasciitilde \texttt{1v} IS ADDED
AND THE CONSTRAINT VA(S(3))<4v IS LEFT UNCHANGED. FOR CASE 2 THE
CONSTRAINT VS(3)=1v IS ADDED AND THE CONSTRAINT VA(S(3))<4v CAN BE
ELIMINATED AS FOLLOWS: VA(S(3))<4v \rightarrow VA[1]<4v \rightarrow v2<4v \rightarrow \texttt{true v}.
CHAPTER VI CONTAINS A COMPLETE DISCUSSION OF THIS PROCESSING. AT
THIS POINT WE WISH ONLY TO OBSERVE THAT MULTIPLE CASES MAY BE
PRODUCED DURING \texttt{\textbackslash vsetv} STATEMENT INTERPRETATION.

WHEN THE INTERPRETER DOES CASE ANALYSIS IT EFFECTIVELY
TRANSLATES THE ORIGINAL PROBLEM INTO MULTIPLE CONSTRAINT
SATISFACTION PROBLEMS. THE DEFINITION OF ANY ONE OF THESE
SUBPROBLEMS IS COMPLETED WHENEVER THE INTERPRETER REACHES \texttt{\textbackslash vendv}
WITH A CONTEXT. THE CONSTRAINT SATISFACTION PROBLEM SOLVING
METHODS CAN THEN EITHER FIND A SOLUTION TO THE PROBLEM REPRESENTED
BY THE CONTEXT OR DETERMINE THAT NO SOLUTION EXISTS. A SOLUTION TO
ANY OF THESE PROBLEMS IS A SOLUTION TO THE ORIGINAL PROBLEM.

THE CRYPT-ADDITION PROBLEM STATED IN REF IN FIGURE II.6B IS
ONE FOR WHICH THIS CASE ANALYSIS IS REQUIRED. EACH TIME THE L3
LOOP IS INTERPRETED A BINARY BRANCH IS REQUIRED AT THE \texttt{\textbackslash vipv}
STATEMENT. HENCE, THERE ARE POTENTIALLY 16 CASES, OR SUBPROBLEMS,
TO BE CONSIDERED. THESE CASES REPRESENT THE 16 POSSIBLE
COMBINATIONS OF CARRY VALUES PRODUCED DURING THE ADDITION PROCESS. 

FIGURE III.1 SHOWS THE CONTEXT AT \textbackslash{VEND} FOR THE CASE WHERE ALL THE CARRY VALUES ARE 0.
CONText

DATA STRUCTURE |
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
</tr>
<tr>
<td>A2</td>
</tr>
<tr>
<td>SUM</td>
</tr>
<tr>
<td>L</td>
</tr>
</tbody>
</table>

VARIABLES
| S(1) | RANGE: 0,1,2,3,4,5,6,7,8,9 |
| S(2) | RANGE: 0,1,2,3,4,5,6,7,8,9 |
| S(3) | RANGE: 0,1,2,3,4,5,6,7,8,9 |
| S(4) | RANGE: 0,1,2,3,4,5,6,7,8,9 |
| S(5) | RANGE: 0,1,2,3,4,5,6,7,8,9 |
| S(6) | RANGE: 0,1,2,3,4,5,6,7,8,9 |
| S(7) | RANGE: 0,1,2,3,4,5,6,7,8,9 |
| S(8) | RANGE: 0,1,2,3,4,5,6,7,8,9 |

CONSTRAINTS
| EXCL(S(1),S(2),S(3),S(4),S(5),S(6),S(7),S(8)) |
| ~(S(7)=0) |
| ~(S(4)=0) |
| S(1)+S(3)<10 |
| S(1)+S(3)=S(8) |
| S(2)+S(5)<10 |
| S(2)+S(5)=S(3) |
| S(3)+S(6)<10 |
| S(3)+S(6)=S(2) |
| S(4)+S(7)<10 |
| S(4)+S(7)=S(6) |
| S(7)=0 |

FIGURE III.1. CASE IN CRYPTO-ADDICTION PROBLEM WHERE CARRY VALUES ARE ALL ZERO
FOR SOME PROBLEMS THE CASE ANALYSIS REQUIRED IS SO GREAT THAT THE MAJOR DIFFICULTY IN FINDING A SOLUTION IS FINDING A CONTEXT WHICH CAN BE INTERPRETED TO \texttt{VEND}. FOR MANY OF THESE PROBLEMS THE INDIVIDUAL SUBPROBLEMS HAVE SUFFICIENTLY STRONG CONSTRAINTS TO MAKE THEIR SOLUTION TRIVIAL. FOR EXAMPLE, IN THE WATER JUG PROBLEM DISCUSSED IN THE PREVIOUS CHAPTER, EACH CONTEXT WILL HAVE A CONSTRAINT OF THE FORM \texttt{V}$S(i)$=\texttt{V} (WHERE I AND J ARE INTEGERS) FOR EACH VARIABLE S(i). THESE CONSTRAINTS ARE ADDED TO THE CONTEXTS DURING INTERPRETATION OF THE COMPUTED \texttt{GOTOV} STATEMENT AT LINE 5 OF THE REF PROCEDURE. FOR SUCH PROBLEMS ARF NEEDS METHODS FOR GUIDING THE INTERPRETER AS TO WHICH CASE TO PURSUE FIRST WHEN CASE ANALYSIS OCCURS, AND ONCE A CASE IS CHOSEN, HOW LONG TO PURSUE THE CASE BEFORE RETURNING TO CONSIDER ANOTHER ONE.

IS AT X2 WILL CONTAIN THE CONSTRAINT $vX[S(1)] = x2v$. IF INTERPRETATION PROCEEDS WITH THIS CASE, THEN THE SECOND VARIABLE IS CREATED AT STATEMENT L1 AND THREE CASES ARE PRODUCED AT THE COMPUTED vGOTOv STATEMENT OF LINE 9. THE CONTEXT WHICH BRANCHES TO vMOVE.boxv WILL CONTAIN THE CONSTRAINT $vS(2) = 3v$. IF INTERPRETATION PROCEEDS WITH THIS CONTEXT, THEN THE THIRD VARIABLE IS CREATED AT STATEMENT vMOVE.boxv. THE FOURTH VARIABLE IS CREATED AT STATEMENT L1, AND THREE NEW CASES ARE PRODUCED AT THE COMPUTED vGOTOv STATEMENT OF LINE 9. THE CONTEXT WHICH BRANCHES TO vCLIMBv WILL CONTAIN THE CONSTRAINT $vS(4) = 2v$. IF INTERPRETATION PROCEEDS WITH THIS CASE, THEN TWO CASES ARE PRODUCED AT THE vIFv STATEMENT OF LINE 11. THE CONTEXT WHICH DOES NOT BRANCH TO vSTEP.DOWNv WILL CONTAIN THE CONSTRAINT $vX[S(3)] = \text{under}\_\text{bananas}$. IF INTERPRETATION PROCEEDS WITH THIS CASE, THEN THE FIFTH VARIABLE IS CREATED AT THE vSETv STATEMENT OF LINE 12 AND TWO CASES ARE PRODUCED AT THE COMPUTED vGOTOv STATEMENT OF LINE 13. THE CONTEXT WHICH BRANCHES TO vGET.bananasv WILL CONTAIN THE CONSTRAINT $vS(5) = 1v$. IF INTERPRETATION PROCEEDS WITH THIS CASE, THEN vENDv IS REACHED AND A SOLUTION ATTEMPT CAN BE MADE ON THE PROBLEM REPRESENTED BY THE CASE. THE CONTEXT HAS THE FOLLOWING FORM AT vENDv:
CON TEXT

DATA STRUCTURE
X
VECTOR: X1, X2, UNDER BANANAS
Y
VECTOR: ON FLOOR, ON BOX
MONKEY
VECTOR: X[S(3)], ON BOX
BOX
VECTOR: X[S(3)], ON FLOOR

VARIABLES
S(1)
RANGE: 1, 2, 3
RANGE: 1, 2, 3
RANGE: 1, 2, 3
RANGE: 1, 2

CONSTRAINTS
X[S(1)] = X2
S(2) = 3
S(4) = 2
X[S(3)] = UNDER BANANAS
S(5) = 1

ARF\*S CONSTRAINT SATISFACTION METHODS HAVE SUFFICIENT POWER TO
SOLVE THE ABOVE PROBLEM WITH ONLY A MODEST AMOUNT OF EFFORT.
HENCE, ARF\*S DIFFICULTY IN FINDING A SOLUTION TO THE MONKEY
PROBLEM IS DECIDING WHICH CASE TO INTERPRET AT EACH BRANCH
STATEMENT.

THE TASK OF GUIDING THE INTERPRETATION WHEN CASE ANALYSIS
OCCURS CAN BE FORMULATED AS A SEARCH PROBLEM TO WHICH HEURISTIC
SEARCH METHODS CAN BE APPLIED. THE OBJECTS OF THIS SEARCH PROBLEM
ARE THE CONTEXT STRUCTURES AND THE OPERATORS ARE THE STATEMENTS IN
THE REF PROCEDURE. THE INITIAL OBJECT IS AN EMPTY CONTEXT AT
\texttt{\texttt{BEGIN}} AND A FINAL OBJECT IS A CONTEXT AT \texttt{VEND} WHOSE CONSTRAINTS HAVE BEEN SATISFIED. THE INTERPRETER IS USED TO APPLY THE OPERATORS TO THE OBJECTS. OPERATOR APPLICATION PRODUCES MORE THAN ONE NEW OBJECT WHEN THE STATEMENT BEING INTERPRETED CAUSES CASE ANALYSIS TO OCCUR.

GIVEN THIS FORMULATION WE MAY CONCLUDE THAT IN GENERAL ARF Translator A PROBLEM STATED IN REF INTO A HEURISTIC SEARCH PROBLEM. THE SPACE IN WHICH THIS SEARCH PROBLEM IS DEFINED HAS THE CHARACTERISTIC THAT EACH OF ITS OBJECTS CONTAINS A BOOLEAN CONSTRAINT SATISFACTION PROBLEM. THE PROBLEM SOLVER MUST HAVE BOTH HEURISTIC SEARCH METHODS FOR CONDUCTING THE SEARCH AND CONSTRAINT SATISFACTION METHODS FOR SOLVING THE PROBLEMS ASSOCIATED WITH THE OBJECTS IN THE SPACE.

3. SIZE OF THE SUBPROBLEM SPACE

THE DIFFICULTY OF THE SEARCH PROBLEM FOR A GIVEN REF PROCEDURE DEPENDS ON THE AMOUNT OF CASE ANALYSIS THE INTERPRETER MUST DO DURING APPLICATION OF THE OPERATORS. THE AMOUNT OF CASE ANALYSIS IS DETERMINED BY TWO FACTORS: THE NUMBER OF BRANCH STATEMENTS INTERPRETED WHICH DEPEND ON THE VALUE OF A \texttt{SELECT} FUNCTION CALL TO DETERMINE THE PATH TAKEN (I.E., WHICH OPERATOR IS APPLICABLE NEXT), AND THE NUMBER OF STATEMENTS INTERPRETED WHICH
contain slot expressions that depend on the value of a select function call for evaluation. For example, the if statement in the procedure defining the crypt-addition problem causes case analysis to occur because evaluation of the branching condition depends on the values of earlier select function calls. Also, in the example which indicated how case analysis can occur during the interpretation of a set statement, it was the slot expression a[s(3)] in the constraint which necessitated the formation of two cases.

In most instances when ARF is given a boolean constraint satisfaction problem stated in REP, no case analysis is needed and the search tree in the subproblem space has only one terminal node. This was the case for the magic square problem discussed above. For such problems there is essentially no search problem in the subproblem space and the primary problem solving burden falls on the constraint satisfaction methods.

For any given process constraint satisfaction problem we may compare the size of ARF's subproblem space with the size of the space in which Floyd's backtracking algorithm searches. For the crypt-addition problem there are \(10^8\) cases for the backtracking algorithm to consider while there are only 16 cases in the subproblem space. This reduction in search allows the constraint
SATISFACTION METHODS TO DO MCST OF THE WORK NECESSARY TO SOLVE THIS PROBLEM.

WE CAN MAKE A SIMILAR COMPARISON OF SEARCH SPACE SIZES FOR A HEURISTIC SEARCH PROBLEM STATED IN REF. IN THIS CASE WE CAN COMPARE THE SIZE OF THE ORIGINAL PROBLEM'S SEARCH SPACE AND THE SIZE OF ARF's SUBPROBLEM SPACE. FOR EXAMPLE, CONSIDER THE MONKEY PROBLEM DISCUSSED IN CHAPTER II. FIGURE III.2A SHOWS THREE LEVELS OF THE SEARCH TREE FORMED IN THE SEARCH SPACE OF THE ORIGINAL PROBLEM. THE NODES OCCUR IN THIS TREE EACH TIME A \texttt{SET} STATEMENT IS ENCOUNTERED WHICH CONTAINS A \texttt{SELECT} FUNCTION CALL, AND THE BRANCHES EMANATING FROM EACH NODE REPRESENT THE POSSIBLE VALUES OF THE \texttt{SELECT} FUNCTION CALL. FIGURE III.2B SHOWS THREE LEVELS OF THE SEARCH TREE FORMED IN THE SUBPROBLEM SPACE. THE NODES OCCUR IN THIS TREE EACH TIME A BRANCH STATEMENT REQUIRES THE CREATION OF NEW CASES, AND THE BRANCHES EMANATING FROM EACH NODE REPRESENT THE CASES CREATED. THE FIGURES ILLUSTRATE THAT THE SUBPROBLEM SEARCH TREE IS THE SMALLER. THE SOLUTION NODE IS AT LEVEL FIVE FOR THESE TREES AND AT THAT LEVEL THERE ARE 83 NODES IN ARF's SUBPROBLEM TREE AND 242 NODES IN THE ORIGINAL PROBLEM'S TREE. SO WE SEE THAT ARF's USE OF VARIABLES HERE TO REPRESENT SELECTIONS HAS TRANSLATED THE ORIGINAL HEURISTIC SEARCH PROBLEM INTO A NEW HEURISTIC SEARCH PROBLEM WITH A SIGNIFICANTLY SMALLER SEARCH SPACE.
Figure III.2A. Three levels of the standard search tree for the monkey problem

Figure III.2B. Three levels of the ARF search tree for the monkey problem
THE SIZE OF THIS REDUCTION FOR ANY GIVEN PROBLEM IS INVERSELY PROPORTIONAL TO THE AMOUNT OF CASE ANALYSIS THE INTERPRETER MUST DO. FOR A PROBLEM LIKE THE WATERJUG PROBLEM STATED IN FIGURE II.8, THE INTERPRETER MUST CREATE EIGHT CASES EACH TIME IT INTERPRETS THE COMPUTED "GOTO" STATEMENT AT LINE 5. THIS CASE PROLIFERATION ESSENTIALLY NULLIFIES THE USE OF VARIABLES AND CAUSES THE SUBPROBLEM SPACE TO BE THE SAME SIZE AS THE ORIGINAL PROBLEM'S SEARCH SPACE. WE HAVE SEEN THAT FOR THE MONKEY PROBLEM AND THE CRYPT-ADDITION PROBLEM A SIGNIFICANT REDUCTION IN SEARCH SPACE SIZE IS OBTAINED, ALTHOUGH THE SEARCH PROBLEM IN THE SUBPROBLEM SPACE REMAINS NONTRIVIAL.

THERE ARE SOME PROCESS CONSTRAINT SATISFACTION PROBLEMS AND HEURISTIC SEARCH PROBLEMS WHICH CAN BE STATED IN REF SO THAT THERE IS ESSENTIALLY NO SEARCH REQUIRED IN THE SUBPROBLEM SPACE. THE MISSIONARIES AND CANNIBALS PROBLEM STATED IN ENGLISH IN FIGURE III.3A IS AN EXAMPLE OF SUCH A HEURISTIC SEARCH PROBLEM. THE ONLY BRANCHEING THAT OCCURS DURING THE INTERPRETATION OF THIS PROBLEM'S REF STATEMENT SHOWN IN FIGURE III.3B IS AT THE "IF" STATEMENT IMMEDIATELY PRECEDING THE "ENDIF" STATEMENT. THIS "IF" STATEMENT FUNCTIONS AS THE TEST FOR THE FINAL STATE AND THE CASE WHICH THE INTERPRETER PRODUCES TO NOT TAKE THE BRANCH TO L1 GOES DIRECTLY TO "ENDIF", THEREBY BECOMING A TERMINAL NODE OF THE SEARCH TREE.
WE HAVE SEEN IN THIS CHAPTER HOW ARF TRANSLATES AND STRUCTURES A PROBLEM STATED IN REF SO THAT PROBLEM SOLVING METHODS CAN BE APPLIED TO IT. THERE ARE TWO BASIC TYPES OF PROBLEMS TO BE SOLVED: THE CONSTRAINT SATISFACTION PROBLEMS CONTAINED IN THE CONTEXT STRUCTURES AND THE HEURISTIC SEARCH PROBLEM OF GUIDING THE INTERPRETER IN THE SUBPROBLEM SPACE. IN CHAPTER IV WE WILL DISCUSS THE SOLVING OF CONSTRAINT SATISFACTION PROBLEMS AND THE METHODS APPLIED TO THE CONTEXT STRUCTURES BY ARF. IN CHAPTER V WE WILL CONSIDER HEURISTIC SEARCH METHODS APPLICABLE TO THE SUBPROBLEM SPACE AND DESCRIBE THE METHODS EMPLOYED BY ARF.
THREE MISSIONARIES AND THREE CANNIBALS WISH TO CROSS A RIVER. THE ONLY MEANS OF CONVEYANCE IS A SMALL BOAT WHICH HAS A CAPACITY OF TWO PEOPLE AND WHICH ALL SIX KNOW HOW TO OPERATE. IF, AT ANY TIME, THERE ARE MORE CANNIBALS THAN MISSIONARIES ON EITHER SIDE OF THE RIVER, THOSE MISSIONARIES WILL BE EATEN BY THE CANNIBALS. HOW CAN ALL SIX GET ACROSS THE RIVER WITHOUT ANY MISSIONARIES BEING EATEN.

FIGURE III.3A. ENGLISH STATEMENT OF THE MISSIONARIES AND CANNIBALS PROBLEM

BEGIN;
    SET MISSIONARIES OF LEFT.SIDE TO 3;
    SET CANNIBALS OF LEFT.SIDE TO 3;
    SET MISSIONARIES OF RIGHT.SIDE TO 0;
    SET CANNIBALS OF RIGHT.SIDE TO 0;
    SET <DEPARTING.SIDE> TO LEFT.SIDE;
    SET <ARRIVING.SIDE> TO RIGHT.SIDE;
L1: SET MISSIONARIES OF BOAT TO SELECT(0,2);
    CONDITION ~(MISSIONARIES OF <DEPARTING.SIDE> < MISSIONARIES OF BOAT);
    SET CANNIBALS OF BOAT TO SELECT(0,2);
    CONDITION ~(CANNIBALS OF <DEPARTING.SIDE> < CANNIBALS OF BOAT);
    CONDITION 0 < MISSIONARIES OF BOAT + CANNIBALS OF BOAT;
    CONDITION ~((2 < MISSIONARIES OF BOAT) + CANNIBALS OF BOAT);
    SET MISSIONARIES OF <DEPARTING.SIDE> TO MISSIONARIES OF
        <DEPARTING.SIDE> - MISSIONARIES OF BOAT;
    SET CANNIBALS OF <DEPARTING.SIDE> TO CANNIBALS OF
        <DEPARTING.SIDE> - CANNIBALS OF BOAT;
    SET MISSIONARIES OF <ARRIVING.SIDE> TO MISSIONARIES OF
        <ARRIVING.SIDE> + MISSIONARIES OF BOAT;
    SET CANNIBALS OF <ARRIVING.SIDE> TO CANNIBALS OF
        <ARRIVING.SIDE> + CANNIBALS OF BOAT;
    CONDITION MISSIONARIES OF LEFT.SIDE = 0 OR ~(MISSIONARIES
        OF LEFT.SIDE < CANNIBALS OF LEFT.SIDE);
    CONDITION MISSIONARIES OF RIGHT.SIDE = 0 OR ~(MISSIONARIES
        OF RIGHT.SIDE < CANNIBALS OF RIGHT.SIDE);
    SET <TEMP> TO <DEPARTING.SIDE>;
    SET <DEPARTING.SIDE> TO <ARRIVING.SIDE>;
    SET <ARRIVING.SIDE> TO <TEMP>;
    IF ~(MISSIONARIES OF RIGHT.SIDE = 3 AND CANNIBALS OF
        RIGHT.SIDE = 3) THEN L1;
END;

FIGURE III.3B. REF STATEMENT OF THE MISSIONARIES AND CANNIBALS PROBLEM
IV. SOLVING CONSTRAINT SATISFACTION PROBLEMS

Let us review the form of the constraint satisfaction problems created during interpretation of a REF procedure. Each problem is represented by a context which contains a list of variables, a list of constraints, and a data structure. Associated with each variable is a list of integers which defines the range of values assignable to the variable. The constraints are in the form of boolean expressions whose truth value can be determined if values are assigned to each variable. The data structure in a context contains the vector elements and attribute-value pairs that have been defined during interpretation. The values in this data structure serve the same function in the statement of the constraint satisfaction problem as do the constraints. That is, a value of 3 for the second element of the vector B is equivalent to the constraint \( \text{v}_B[2] = 3 \). Such constraints are represented in the data structure rather than as part of the constraint list in order to save processing time and storage space during both creation and solution of the problem.

A. DESIGNING A CONSTRAINT SATISFIER

We will now present a progression of designs for a problem solver to solve problems of this form and discuss the relative
MERITS OF EACH DESIGN. THIS PRESENTATION WILL PROVIDE THE FRAMEWORK FOR OUR DESCRIPTION OF THE CONSTRAINT SATISFACTION METHODS IN THE CURRENT ARF.

1. VALUE ASSIGNMENT METHODS

FIRST OF ALL, OBSERVE THAT IT IS POSSIBLE TO ENUMERATE ALL POSSIBLE SETS OF VALUES FOR THE VARIABLES SINCE EACH PROBLEM CONTAINS A FINITE NUMBER OF VARIABLES EACH HAVING A FINITE SET OF POSSIBLE VALUES. HENCE, ONE COULD ORGANIZE A PROBLEM SOLVER TO SYSTEMATICALLY GENERATE EACH POSSIBLE SET OF VARIABLE VALUES AND TEST IF THE SET CONSTITUTES A SOLUTION BY EVALUATING EACH OF THE CONSTRAINTS. FOR EXAMPLE, CONSIDER THE SUBPROBLEM OF THE CRYPT-ADDITION PROBLEM IN WHICH ALL THE CARRY VALUES ARE 0. THE CONTEXT WHICH REPRESENTS THIS SUBPROBLEM IS THE ONE SHOWN IN FIGURE III.1. LET AN 8-TUPLE REPRESENT A SET OF VALUES FOR THIS PROBLEM'S EIGHT VARIABLES. THEN A PROBLEM SOLVER MIGHT CONSIDER ALL POSSIBLE SETS BY GENERATING THEM IN LEXICOGRAPHIC ORDER AS FOLLOWS:

(0,0,0,0,0,0,0,0)  
(0,0,0,0,0,0,1,0)  
(0,0,0,0,0,0,2,0)  
  
(9,9,9,9,9,9,9,8)  
(9,9,9,9,9,9,9,9)
There are 10^18 such sets for this problem, and since the problem has no solution they must all be generated.

A much more efficient way of considering all possible solutions is to use a backtracking method. Backtracking achieves its effectiveness by requiring that the problem solver be able to test a partially generated solution. That is, for the problems we are considering instead of generating an entire set of values and then testing if the set is a solution, the backtracking method generates each value of a set separately and tests the partially defined set after each value is generated. The test can be implemented as a reevaluation of each of the constraints with the values in the partially generated set assigned to the appropriate variables. If a constraint reduces to "false" during the reevaluation, then the problem solver may discard the partial set and remove from consideration all possible completions of the discarded set.

Consider the use of this form of backtracking on the crypt-addition subproblem discussed above. Assume that the partial sets are being created by first generating a value for S(1), then a value for S(2), etc., and that the sets are being generated in lexicographic order as described above. As generation proceeds for the subproblem under consideration, the partial set (0,0,...)
WILL CAUSE THE CONSTRAINT $\forall \text{excl}(S(1), S(2), S(3), S(4), S(5), S(6), S(7), S(8)) \forall$ TO BECOME FALSE. THIS PARTIAL SET CAN BE DISCARDED AND GENERATION CAN PROCEED WITH $(0, 1, \ldots)$. THE SAME CONSTRAINT DISQUALIFIES $(0, 1, 0, \ldots)$, $(0, 1, 1, \ldots)$, $(0, 1, 2, 0, \ldots)$, ETC. AS GENERATION PROCEEDS THE NEXT CONSTRAINT TO DISQUALIFY PARTIAL SETS IS $\forall S(2) + S(5) = S(3) \forall$. THE FIRST PARTIAL SET CONTAINING FIVE VALUES WHICH IT AND THE $\forall \text{excl} \forall$ CONSTRAINT DO NOT DISQUALIFY IS $(0, 1, 3, 4, 2, \ldots)$. THE GENERATION CONTINUES IN THIS FASHION UNTIL ALL PARTIAL SETS HAVE BEEN DISCARDED AND THE CONCLUSION IS REACHED THAT NO SOLUTION EXISTS FOR THE SUBPROBLEM. THE NUMBER OF PARTIAL SETS WHICH MUST BE GENERATED TO REACH THIS CONCLUSION IS APPROXIMATELY 58,000.

NOTE THAT WHEN A PARTIAL SET HAVING TWO ELEMENTS IS DISCARDED, SUCH AS $(9, \ldots)$, $10^6$ COMPLETE SETS ARE EXCLUDED FROM CONSIDERATION AS SOLUTIONS; WHEREAS WHEN A PARTIAL SET HAVING FIVE ELEMENTS IS DISCARDED, SUCH AS $(0, 1, 2, 3, 4, \ldots)$, ONLY $10^3$ COMPLETE SETS ARE EXCLUDED. THIS OBSERVATION MOTIVATES ONE TO DESIGN SCHEMES THAT WILL ENABLE THE PROBLEM SOLVER TO DISCARD PARTIAL SETS HAVING AS FEW ELEMENTS AS POSSIBLE. ONE STRATEGY FOR SUCH A SCHEME IS TO ALLOW THE PROBLEM SOLVER TO DECIDE FOR EACH PROBLEM THE ORDER IN WHICH ELEMENTS ARE ASSIGNED VALUES IN THE SETS BEING GENERATED. THAT IS, THE PROBLEM SOLVER DESCRIBED ABOVE ALWAYS GENERATES A VALUE FOR $S(1)$ FIRST, THEN A VALUE FOR $S(2)$,
ETC., BUT THIS NEED NOT BE THE CASE. FOR EXAMPLE, THE PROBLEM
SOLVER MIGHT CHOOSE TO BEGIN BY GENERATING VALUES FOR S(7) RATHER
THAN FOR S(1). SINCE THE SUBPROBLEM CONTAINS THE CONTRADICTORY
CONSTRAINTS \( \forall S(7) = 0 \) AND \( \forall S(7) = 0 \), IT WOULD BE NECESSARY TO
GENERATE ONLY TEN PARTIAL SETS TO DETERMINE THAT THE PROBLEM HAS
NO SOLUTION, I.E. \( (\ldots, 0), (\ldots, 1), \ldots, (\ldots, 9) \).

ONE WAY FOR THE PROBLEM SOLVER TO DETERMINE A GENERATION
ORDER IS TO EXAMINE THE CONTENT AND FORM OF A PROBLEM'S
CONSTRAINTS. FOR EXAMPLE, ARF USES A METHOD WHICH ORDERS THE
GENERATION BASED ON AN ATTEMPT TO MAKE AS MANY CONSTRAINTS
REDUCIBLE TO \( \forall \text{true} \) OR \( \forall \text{false} \) AS SOON IN THE GENERATION AS
POSSIBLE WITH PREFERENCE GIVEN TO THOSE CONSTRAINTS WHICH MOST
RESTRICT THE VARIABLE VALUES. THIS METHOD DECIDES WHICH VARIABLE
TO ASSIGN A VALUE NEXT BY SELECTING THOSE CONSTRAINTS WHICH HAVE
THE LEAST NUMBER OF UNVALUED VARIABLES. THESE ARE THE CONSTRAINTS
WHICH ARE REDUCIBLE TO \( \forall \text{true} \) OR \( \forall \text{false} \) WITH THE LEAST NUMBER OF
ADDITIONAL GENERATIONS. IT THEN ATTEMPTS TO SELECT A CONSTRAINT
FROM THIS SET WHICH MOST RESTRICTS THE VALUES OF THE VARIABLES
OCcurring IN IT. THIS JUDGMENT IS BASED ON THE FORM OF THE
CONSTRAINT. THAT IS, THERE IS DEFINED AN OPERATOR ORDERING AS
FOLLOWS: \( =, <, \text{excl}, \text{~excl}, \text{~=}, \) AND \( \forall \); CONSTRAINTS ARE ORDERED BY
THEIR MAIN OPERATORS USING THIS ORDERING SO THAT EQUATIONS ARE
ASSUMED TO BE MOST RESTRICTIVE AND DISJUNCTIONS LEAST RESTRICTIVE.
CONSTRAINTS SELECTED

VARIES CHosen FOR GENERATION

\[-(S(7) = 0)\]
\[-(S(4) = 3)\]
\[S(7) = 0\] * \[S(7)\]
\[-(S(4) = 0)\]
\[S(4) + S(7) < 10\] * \[S(4)\]
\[S(4) + S(7) = S(6)\] * \[S(6)\]
\[S(3) + S(6) < 10\] * \[S(3)\]
\[S(1) + S(3) < 10\]
\[S(3) + S(6) = S(2)\] * \[S(2)\]
\[S(1) + S(3) < 10\]
\[S(2) + S(5) < 10\]
\[S(2) + S(5) = S(3)\] * \[S(5)\]
\[S(1) + S(3) < 10\] * \[S(1)\]
\[S(1) + S(3) = S(8)\] * \[S(1)\]
\[EXCL S(1), S(2), S(3), S(4), S(5), S(6), S(7), S(8)\]
\[S(8)\]

* INDICATES MOST RESTRICTIVE CONSTRAINT IN THE SET

FIGURE IV.1. ORDERING OF VARIABLES FOR GENERATION IN THE EXAMPLE CASE FROM THE CRYPTO-ADDITION PROBLEM
AS WE OBSERVED EARLIER THE CHOICE OF S(7) TO BE GENERATED FIRST IS THE MOST DESIRABLE FOR THIS PROBLEM. TO FURTHER SEE THE POWER OF THE ORDERING DETERMINED BY THE ALGORITHM ASSUME THAT THE PROBLEM DOES NOT HAVE THE CONSTRAINT \( vS(7) = 0 \). THE SAME ORDERING WOULD HAVE BEEN DETERMINED. DURING THE GENERATION, ACCEPTABLE VALUES FOR S(7) AND S(4) COULD BE FOUND, BUT THE ZERO VALUE OF S(7) WOULD PREVENT ANY VALUE FOR S(6) FROM SATISFYING BOTH THE \( \text{EXCLV} \) CONSTRAINT AND THE CONSTRAINT \( vS(4) + S(7) = S(6) \). HENCE, ONLY NINETY-TWO PARTIAL SETS WOULD BE GENERATED, EACH HAVING THREE OR LESS ELEMENTS.

IF WE ASSUME THAT THE PROBLEM HAS THE CONSTRAINT \( vS(7) = 0 \) BUT DOES NOT HAVE THE CONSTRAINT \( vS(7) = 0 \), THEN THE ORDERING IS NOT AS EFFECTIVE AS IN THE OTHER TWO CASES. THE SAME ORDERING WOULD BE DETERMINED. IN THIS CASE THE CONTRADICTION COMES FROM THE CONSTRAINTS \( vS(3) + S(6) = S(2) \) AND \( vS(2) + S(5) = S(3) \). S(5) IS THE SIXTH VARIABLE GIVEN A VALUE IN EACH SET SO THAT APPROXIMATELY 18,000 PARTIAL SETS HAVING SIX OR LESS ELEMENTS WOULD BE GENERATED. THE WEAKNESS OF THE ORDERING FOR THIS CASE REFLECTS THE FACT THAT THE ORDERING ALGORITHM GIVES PREFERENCE TO ENABLING THE EVALUATION OF THE MAXIMUM NUMBER OF CONSTRAINTS WITH ONLY A SECONDARY CONSIDERATION GIVEN TO THE RESTRICTIVENESS OF THE CONSTRAINT. A MORE SOPHISTICATED ORDERING ALGORITHM MIGHT OBSERVE
That the two contradictory constraints are equations and that they contain two variables in common, thus greatly restricting the values of those two variables. Such an observation would lead to the early generation of values for the four variables needed to evaluate the two constraints.

2. Constraint Manipulation Methods

If we consider ways to further improve the power of this solution procedure for constraint satisfaction problems, we might observe that there is much information in the constraints that is not being utilized. For example, in the subproblem we have been considering, all ten possible values for \( s(7) \) will be generated even though the constraint \( \neg s(7) = 0 \) clearly indicates what the value of \( s(7) \) must be. Similarly, if values have been generated for \( s(7) \) and \( s(4) \), then the constraint \( \neg s(4) \land s(7) = s(6) \) excludes all but one value for \( s(6) \).

One may include in a problem solver routines which perform algebraic manipulations on constraints with the goal of reducing the amount of generation required during the backtracking search or, as in this case, eliminating the need for any search at all. These routines perform three principle functions: elimination of variables from constraints, elimination of elements from variable
RANGES, AND DEDUCTION OF INCONSISTENCIES AMONG THE CONSTRAINTS. CONSIDER HOW SUCH ROUTINES COULD PERFORM THESE FUNCTIONS IN THE CRYPT-ADDITION SUBPROBLEM WE HAVE BEEN USING AS AN EXAMPLE.

EQUALITY CONSTRAINTS SUCH AS \( vS(7)=0 \) OR \( vS(1)+S(3)=S(8) \) CAN BE USED TO ELIMINATE A VARIABLE BY SUBSTITUTING ALL OCCURRENCES OF THE VARIABLE BY AN EQUIVALENT EXPRESSION. THE EQUIVALENT EXPRESSION CAN BE STORED IN THE CONTEXT STRUCTURE AS THE VALUE OF THE VARIABLE AND ALL OCCURRENCES OF THE VARIABLE IN VECTOR ELEMENTS, ATTRIBUTE VALUES, OR VALUES OF OTHER VARIABLES CAN BE REPLACED BY THE EQUIVALENT EXPRESSION. HENCE, IN THE EXAMPLE SUBPROBLEM ALL OCCURRENCES OF \( S(7) \) WOULD BE REPLACED BY 0 AND ALL OCCURRENCES OF \( S(8) \) WOULD BE REPLACED BY \( vS(1)+S(3) \).

CONSTRAINTS SUCH AS \( vS(7)\neq 0 \) AND \( vS(4)<9 \) CAN BE USED TO ELIMINATE ELEMENTS FROM THE RANGE OF A VARIABLE. AFTER A CONSTRAINT HAS BEEN USED IN THIS WAY IT MAY BE DELETED FROM THE CONTEXT IF IT IS TRUE FOR ALL REMAINING RANGE VALUES. THIS IS THE CASE FOR BOTH \( vS(7)\neq 0 \) AND \( vS(4)<9 \). IF A CONSTRAINT CAUSES THE RANGE OF A VARIABLE TO BE REDUCED TO ONE ELEMENT, THEN THAT ELEMENT CAN BE SET AS THE VALUE OF THE VARIABLE IN THE CONTEXT.

INCONSISTENCIES MAY BE DEDUCED FROM THE CONSTRAINTS IN SEVERAL WAYS: A CONSTRAINT MAY REDUCE TO FALSE FOLLOWING THE ELIMINATION OF A VARIABLE AND REAPPLICATION OF SIMPLIFICATION
Routines, a constraint may cause removal of the entire range of a variable, or combinations of constraints such as vs(7)=0 v and vs(7)~0 v may be inconsistent.

We see that by appealing directly to the constraints the search for possible solutions may be reduced or eliminated. The effectiveness of these constraint manipulation methods is related to how and when they are applied, just as the effectiveness of the backtracking algorithm is related to the order in which the values are generated. A simple way of organizing the problem solver is to apply all the constraint manipulation methods to each constraint in the order in which they occur in the problem. In our example subproblem this would mean consideration of the vs excl v constraint first, then vs(7)~0 v, then vs(4)~0 v, etc. In this problem the most obvious inconsistency occurs between the constraints vs(7)=0 v and vs(7)~0 v, but if the problem solver does not consider altering the order in which it processes constraints then this inconsistency will not be recognized until all twelve constraints have been processed.

This implies that the effectiveness of the problem solver can again be increased by designing an algorithm for selecting the order in which constraints are processed. This ordering should be designed so that those constraints will be considered first for
WHICH THERE IS A HIGH PROBABILITY THAT A VARIABLE CAN BE
ELIMINATED, VARIABLE RANGE ELEMENTS CAN BE DELETED, OR AN
INCONSISTENCY CAN BE DETERMINED. THE NATURE OF THE ORDERING
ALGORITHM SHOULD DEPEND ON THE CONSTRAINT MANIPULATION METHODS
WHICH ARE AVAILABLE. A SIMPLE SCHEME WHICH IS SUITABLE FOR USE IN
ARF GROUPS ALL CONSTRAINTS HAVING OCCURRENCES OF EXACTLY ONE
VARIABLE FIRST, ALL EQUATIONS NEXT, \( \forall \text{EXCL}\) CONSTRAINTS NEXT, AND
ALL OTHERS FOLLOWING. THE ORDERING WITHIN EACH OF THESE GROUPS
(EXCEPT THE FIRST) IS BASED ON THE NUMBER OF VARIABLES OCCURRING
IN THE CONSTRAINT, THOSE HAVING THE FEWEST VARIABLES BEING
CONSIDERED FIRST.

FOR OUR EXAMPLE SUBPROBLEM THIS ORDERING WOULD DICTATE THAT
CONSTRAINTS \( \forall S(7)=0 \), \( \forall S(4)=0 \) AND \( \forall S(7)=0 \) BE CONSIDERED FIRST.
HENCE, THE SUBPROBLEM COULD BE ELIMINATED AFTER CONSIDERATION OF
AT MOST THREE CONSTRAINTS WITH NO BACKTRACKING REQUIRED. TO
FURTHER ILLUSTRATE THE MERIT OF THIS ORDERING ASSUME THAT THE
PROBLEM DOES NOT CONTAIN THE CONSTRAINT \( \forall S(7)=0 \). IF THE PROBLEM
SOLVER USED \( \forall S(7)=0 \) TO REPLACE ALL OCCURRENCES OF \( S(7) \) IN THE
CONSTRAINTS BY 0, THEN THE CONSTRAINT \( \forall S(4)+S(7)<10 \) WOULD BECOME
\( \forall S(4)<10 \), CONSTRAINT \( \forall S(4)+S(7)=S(6) \) WOULD BECOME \( \forall S(4)=S(6) \),
AND THE \( \forall \text{EXCL} \) CONSTRAINT WOULD BECOME \( \forall \text{EXCL} (S(1),S(2),S(3),
S(4),S(5),S(6),0,S(8)) \). THE ORDERING WOULD DICTATE THAT \( \forall S(4)<10 \)
BE CONSIDERED NEXT SINCE IT HAS ONLY ONE VARIABLE. THE ORDERING


ABLE TO USE THESE NEW CONSTRAINTS TO MAKE DEDUCTIONS WHICH WILL FURTHER REDUCE THE GENERATION REQUIRED TO COMPLETE THE CURRENT PARTIAL SET. HENCE, THEY CAN PROFITABLY BE APPLIED TO EACH NEW CONSTRAINT AS THE SEARCH PROGRESSES.

IF CONSTRAINT MANIPULATION CONTINUES DURING THE BACKTRACKING IN THIS MANNER, THEN THE ISSUE OF WHAT ORDER THE VARIABLES ARE TO BE ASSIGNED VALUES BY THE BACKTRACKING ROUTINE CAN BE RAISED ANEW. THAT IS, THE ORDERING ROUTINE MAY HAVE DICTATED THAT S(I) BE ASSIGNED A VALUE FIRST AND S(J) BE ASSIGNED A VALUE SECOND. BUT THE ASSIGNMENT OF A VALUE TO S(I) CREATES A NEW SET OF CONSTRAINTS AND S(J) MAY NO LONGER BE THE MOST DESIRABLE NEXT VARIABLE FOR VALUE ASSIGNMENT. THIS IMPLIES THE ORDERING ROUTINE COULD BE SIMPLIFIED SO THAT IT OUTPUTS ONLY THE NEXT VARIABLE TO BE ASSIGNED A VALUE RATHER THAN AN ORDERING OF ALL THE VARIABLES IN THE PROBLEM.

HENCE, THE FIRST PROPOSAL IS FOR ANOTHER DEGREE OF FLEXIBILITY IN THE SOLVING PROCESS; NAMELY, AFTER EACH VALUE ASSIGNMENT DURING THE BACKTRACKING SEARCH THE CONSTRAINT MANIPULATION ROUTINES CAN BE APPLIED TO THE NEW CONSTRAINTS AND THEN THE DECISION MADE AS TO WHICH VARIABLE IS TO BE ASSIGNED A VALUE NEXT.

THE SECOND PROPOSAL FOR INTERMIXING THE CONSTRAINT
MANIPULATION AND BACKTRACKING PHASES CONCERN THE PROCESSING TIME REQUIRED BY THE TWO PHASES. WE OBSERVED EARLIER THAT THE PRIMARY PURPOSE FOR APPLYING THE CONSTRAINT MANIPULATION METHODS IS TO ELIMINATE THE NEED FOR OR REDUCE THE PROCESSING REQUIRED BY THE BACKTRACKING SEARCH. HENCE, WE MUST TAKE CARE THAT THE PROCESSING TIME REQUIRED BY THE CONSTRAINT MANIPULATION DOES NOT EXCEED THE REDUCTION IN PROCESSING TIME OBTAINED FOR THE BACKTRACKING SEARCH. THAT IS, IF A PROBLEM HAS A SMALL SOLUTION SPACE AND/OR FEW CONSTRAINTS, THE PROCESSING COST OF THE BACKTRACKING SEARCH MAY BE SMALL ENOUGH TO MAKE UNPROFITABLE THE CONSTRAINT MANIPULATION.


WHENEVER A CONSTRAINT IS SELECTED BY THE ORDERING ALGORITHM TO BE CONSIDERED BY THE CONSTRAINT MANIPULATION METHODS, THE
ESTIMATES WOULD BE COMPUTED WITH RESPECT TO THAT CONSTRAINT. IF
THE ESTIMATED COST OF APPLYING THE CONSTRAINT MANIPULATION METHODS
TO THE CONSTRAINT EXCEEDS SOME PORTION OF THE ESTIMATED COST OF
DOING THE BACKTRACKING SEARCH, THE CONSTRAINT WOULD BE MARKED
PROCESSED WITH NO APPLICATION OF THE CONSTRAINT MANIPULATION
METHODS. HENCE, THE MANIPULATION METHODS WOULD BE APPLIED ONLY IF
THERE WAS A GOOD CHANCE THAT THE COST OF THEIR APPLICATION WOULD
BE LESS THAN THE RESULTING REDUCTION IN THE COST OF THE
BACKTRACKING SEARCH.

NOTE THAT THIS DECISION PROCESS SHOULD CONTINUE DURING THE
BACKTRACKING SEARCH AS NEW CONSTRAINTS ARE PRESENTED TO THE
CONSTRAINT MANIPULATION METHODS. IN THESE CASES THE SEARCH COST
ESTIMATE WOULD BE AN ESTIMATE OF THE REMAINING SEARCH TO BE DONE
GIVEN THE VARIABLE VALUES ALREADY SET IN THE PROBLEM.

4. THE RESULTING DESIGN

THIS COMPLETES OUR PROGRESSION THROUGH THE DESIGN OF A
PROBLEM SOLVER FOR SOLVING CONSTRAINT SATISFACTION PROBLEMS. THE
SUGGESTED DESIGN IS NOT COMPLETE IN THAT WE HAVE NOT DISCUSSED
PARTICULAR METHODS FOR CONSTRAINT MANIPULATION, EXPRESSION
EVALUATION AND SIMPLIFICATION, ETC. WHAT WE HAVE IS A TOP LEVEL
DESIGN OR STRATEGY FOR ATTACKING THE PROBLEMS.
THE FLOW CHARTS OF FIGURES IV.2A AND IV.2B INDICATE A PROBLEM SOLVER WHICH INCORPORATES THE DESIGN WE HAVE DISCUSSED. IN THE FLOW CHARTS A PROBLEM IS ASSUMED TO HAVE A LIST OF PROCESSED CONSTRAINTS AND A LIST OF UNPROCESSED CONSTRAINTS. WHEN A PROBLEM IS INPUT TO THE PROBLEM SOLVER ALL OF ITS CONSTRAINTS ARE ON THE UNPROCESSED LIST; ALSO, WHENEVER A NEW CONSTRAINT IS PRODUCED DURING THE APPLICATION OF THE CONSTRAINT MANIPULATION METHODS OR THE ASSIGNMENT OF A VALUE TO A VARIABLE, IT IS PUT ON THE UNPROCESSED LIST. AS THE PROBLEM SOLVING PROCEEDS, EACH UNPROCESSED CONSTRAINT IS CONSIDERED AND MOVED TO THE PROCESSED LIST.
BEGIN

Yes

Have all constraints been processed?

No

Select an unprocessed constraint to be processed. Remove it from the unprocessed list.

CE ← cost estimate for processing the selected constraint.

SE ← cost estimate for backtracking.

CE > DELTA * SE?

No

Process the selected constraint.

Failed

Succeeded

Yes

Put the selected constraint on the processed list.

Figure IV.2A. Executive for Constraint Satisfier
Have all variables been assigned values?

Select a variable to be assigned a value.

Make a copy of the problem. Record in the copy which variable was selected. Delete the first element of the selected variable's range in the copy. Store the copy in the top of the stack.

Assign the first range element of the selected variable as its value.

Any problems in the stack?

Remove the top problem from the stack and consider it as the current problem.

Does the range of the selected variable have only 1 element?

Solution found

Succeeded

Failed

Figure IV.2B. Executive for Constraint Satisfier (continued)
FIGURE IV.2A SHOWS THE APPLICATION OF THE CONSTRAINT MANIPULATION METHODS USING BOTH AN ORDERING ALGORITHM TO SELECT EACH CONSTRAINT FOR PROCESSING AND AN ESTIMATING FACILITY FOR DECIDING IF THE MANIPULATION METHODS CAN BE PROFITABLY APPLIED. IF WHEN ALL THE CONSTRAINTS HAVE BEEN CONSIDERED FOR PROCESSING THE PROBLEM IS STILL NOT SOLVED, THEN CONTROL MOVES TO FIGURE IV.2B WHERE A STEP IN THE BACKTRACKING SEARCH IS TAKEN. PROBLEM COPIES ARE KEPT IN A STACK SO THAT BACKTRACKING CAN OCCUR WHEN NECESSARY. AT EACH STEP IN THE SEARCH AN ORDERING ROUTINE IS USED TO SELECT THE NEXT VARIABLE TO BE ASSIGNED A VALUE AND IF THE ASSIGNMENT IS SUCCESSFULLY MADE THEN THE MANIPULATION METHODS ARE APPLIED TO ANY NEW CONSTRAINTS.

B. ARPS CONSTRAINT SATISFACTION METHODS

MUCH OF THE DESIGN DISCUSSED ABOVE EVOLVED AS A DIRECT RESULT OF OUR EXPERIENCE IN BUILDING ARP AND OBSERVING ITS BEHAVIOR. THE PART OF ARP WHICH SOLVES THE CONSTRAINT SATISFACTION SUBPROBLEMS REPRESENTED AS CONTEXT STRUCTURES HAS AN ORGANIZATION SIMILAR TO THAT PRESENTED IN FIGURE IV.2, BUT IT DOES NOT YET CONTAIN ALL OF THE FEATURES INCLUDED IN THAT PRESENTATION. IN PARTICULAR, IT DOES NOT USE AN ORDERING MECHANISM FOR SELECTING WHICH CONSTRAINT TO PROCESS NEXT, AND IT DOES NOT CONTAIN THE FACILITIES FOR
ESTIMATING THE COST OF CONSTRAINT MANIPULATIONS AND BACKTRACKING.

THE REASON WHY ARF DOES NOT USE AN ORDERING MECHANISM FOR SELECTING UNPROCESSED CONSTRAINTS IS BASED ON ISSUES WHICH ARISE IN CONDUCTING THE SEARCH IN THE SUBPROBLEM SPACE. THE HEURISTIC SEARCH METHODS (TO BE DISCUSSED IN CHAPTER V) ASSUME THAT ALL CONSTRAINTS ARE PROCESSED AT THE TIME THEY ARE ADDED TO A CONTEXT. HENCE, ARF DOES NOT MAINTAIN A LIST OF UNPROCESSED CONSTRAINTS IN EACH CONTEXT AS INTERPRETATION PROCEEDS, BUT PROCESSES EACH CONSTRAINT AT THE TIME IT IS CREATED. THE BACKTRACKING SEARCH IS BEGUN WHEN A CONTEXT IS INTERPRETED TO \( \text{VEND} \) AND PROCEEDS AS IN FIGURE IV.2B WITH NEW CONSTRAINTS CONTINUING TO BE PROCESSED AT THE TIME THEY ARE CREATED. THE ADVISABILITY OF THIS ORGANIZATION WILL BE CONSIDERED FURTHER IN CHAPTER V AS PART OF THE DISCUSSION OF THE SEARCH PROBLEM IN THE SUBPROBLEM SPACE.

WE NOW DESCRIBE ARF'S FACILITIES FOR EVALUATING EXPRESSIONS, ENTERING VALUES INTO CONTEXTS, AND PROCESSING CONSTRAINTS. THESE ROUTINES ASSUME THAT AN EXPRESSION IS REPRESENTED INTERNALLY AS A LIST STRUCTURE CONTAINING AN OPERATOR AND A LIST OF OPERANDS. THE OPERATORS CURRENTLY DEFINED IN ARF ARE \( \text{VINT} \), \( \text{VSYM} \), =, <, >, +, -, \( \text{VOF} \), \( \text{VELEM} \), \( \text{VELE} \), \( \text{VECL} \), \( \text{AND} \), AND \( \text{OR} \). AN \( \text{VINT} \) EXPRESSION REPRESENTS AN INTEGER CONSTANT AND HAS AS ITS OPERAND AN INTEGER DATA TERM. A \( \text{VSYM} \) EXPRESSION REPRESENTS A REF IDENTIFIER AND HAS
AS ITS OPERAND THE NAME OF A STRUCTURE WHICH CONTAINS THE NAME OF
THE IDENTIFIER. THE OPERANDS OF EACH OF THE OTHER EXPRESSION TYPES
ARE THEMSELVES EXPRESSIONS. AN \texttt{OF} EXPRESSION HAS TWO OPERANDS, \texttt{X}
AND \texttt{Y}, AND INDICATES THE VALUE OF THE ATTRIBUTE \texttt{X} OF IDENTIFIER \texttt{Y}.
AN \texttt{ELEM} EXPRESSION HAS TWO OPERANDS, \texttt{X} AND \texttt{I}, AND INDICATES THE
VALUE IN THE ITH ELEMENT OF THE VECTOR \texttt{X}. A \texttt{SELECT} EXPRESSION HAS
TWO INTERPRETATIONS; WHEN IT APPEARS IN A REF STATEMENT IT
REPRESENTS A CALL ON THE \texttt{SELECT} FUNCTION AND HAS TWO OPERANDS;
WHEN IT OCCURS IN AN EXPRESSION IN A CONTEXT STRUCTURE IT
REPRESENTS AN INTERPRETER CREATED VARIABLE AND HAS A SINGLE
INTEGER OPERAND WHICH IDENTIFIES THE VARIABLE. PLUS EXPRESSIONS
MAY HAVE ANY NUMBER OF OPERANDS WHILE MINUS EXPRESSIONS ARE
RESTRICTED TO ONE OPERAND.

WE MAY ILLUSTRATE THE INTERNAL FORM OF AN EXPRESSION BY USING
A PREFIX NOTATION IN WHICH AN EXPRESSION IS DENOTED BY ITS
OPERATOR FOLLOWED BY ITS LIST OF OPERANDS SEPARATED BY COMMAS AND
ENCLOSED IN PARENTHESES. USING THIS NOTATION THE INTERNAL FORM OF
THE EXPRESSION \texttt{VM(1) - B OF C + 4 = 15} IS AS FOLLOWS:

$$= (+ (\texttt{ELEM(SYM(B)}, \texttt{INTE(1)}), - (\texttt{OF(SYM(B)}, \texttt{SYM(C)})),
\texttt{INTE(4)}), \texttt{INTE(15)})$$
1. EXPRESSION EVALUATION AND SIMPLIFICATION

There is a single routine in ARF which is called whenever a REF expression is to be evaluated in a context. This routine acts as an executive for a set of other routines called actions which actually do the manipulations and substitutions during the evaluation. These action routines are organized in lists indexed by expression operators; that is, there is a list of actions for equations, a list of actions for sums, a list for negations, etc. The executive uses a simple bottom-up algorithm which recursively evaluates each of the subexpressions of an expression before evaluating the expression itself. Its flow of control is shown in Figure IV.3. Each action routine exits + or - to indicate if evaluation of the expression input to the action is complete. The executive responds to a - exit by ignoring the remaining actions on the list.
Figure IV.3. ARF's expression evaluation executive

$M[1] + -(B \text{ OF } C) + 4 = M[1]$

$M[1] + -(B \text{ OF } C) + 4$

$M[1]$

$M + M$

$1 + 1$


$-(B \text{ OF } C)$

$B \text{ OF } C$

$B + B$

$C + C$

$B \text{ OF } C + S(2)$ (R136)

$-S(2) + -S(2)$

$4 + 4$


$M[1]$

$M + M$

$1 + 1$


$(4 + M[1] + -S(2)) + -(M[1]) + 4 + M[1] + -S(2) + -(M[1])$ (R35)

$4 + M[1] + -S(2) + -(M[1]) + 4 + -S(2)$ (R15)

$4 + -S(2) = 0 + -S(2) = -4$ (R16)

$-S(2) = -4 + S(2) = 4$ (R50)

**FIGURE IV.4A. EXAMPLE OF ARPVS EXPRESSION EVALUATION**
EVALUATION OF $v\neg\neg(-M[1] < 2) v$ IN A CONTEXT WHOSE DATA STRUCTURE CONTAINS THE VALUE $-S(1)$ FOR $M[1]$.

$$
\neg(-M[1] < 2)
\neg M[1] < 2
\neg M[1]
M[1] \\
M \rightarrow M
1 \rightarrow 1
M[1] \rightarrow -S(1) \quad \text{(R137)}
\neg(-S(1)) \rightarrow S(1) \quad \text{(R116)}

2 \rightarrow 2
S(1) < 2 \rightarrow S(1) < 2
\neg(S(1) < 2) \rightarrow 2 < 1 + S(1) \quad \text{(R7)}
1 + S(1) \rightarrow 1 + S(1)
2 < 1 + S(1) \rightarrow 1 < S(1) \quad \text{(R53)}
$$

FIGURE IV.4E. EXAMPLE OF ARFVS EXPRESSION EVALUATION
2. ENTERING A VALUE INTO A CONTEXT

WE NOW DESCRIBE THE ROUTINE WHICH IS CALLED WHENEVER A VARIABLE, VECTOR ELEMENT, OR IDENTIFIER-ATTRIBUTE PAIR IS TO BE ASSIGNED AN EXPRESSION AS A VALUE IN A CONTEXT. (NOTE, THE EXPRESSION MAY BE AN \texttt{VINTEV} OR \texttt{VSYM} EXPRESSION AND THEREFORE DENOTE A CONSTANT). THIS ROUTINE APPROPRIATELY ENTERS THE VALUE EXPRESSION INTO THE CONTEXT. IF A VALUE ALREADY EXISTS FOR THE VECTOR ELEMENT, VARIABLE, OR IDENTIFIER-ATTRIBUTE PAIR, THEN THE ROUTINE CREATES A NEW CONSTRAINT WHICH EQUATES THE OLD AND NEW VALUES. IN ADDITION, THE ROUTINE REPLACES ALL REFERENCES IN VALUE EXPRESSIONS AND CONSTRAINTS TO THE VECTOR ELEMENT, VARIABLE, OR IDENTIFIER-ATTRIBUTE PAIR BY THE NEW VALUE. ANY NEW CONSTRAINTS PRODUCED DURING THIS SUBSTITUTION PROCEDURE ARE REPROCESSED BY THE CONSTRAINT MANIPULATION ROUTINES.

IN THE CASE WHERE AN EXPRESSION WHICH IS NOT AN \texttt{VINTEV} EXPRESSION IS BEING ASSIGNED AS THE VALUE OF A VARIABLE, THE ROUTINE CREATES TWO NEW CONSTRAINTS TO BE ADDED TO THE CONTEXT. THESE CONSTRAINTS ORIGINATE FROM THE FACT THAT THE VALUE EXPRESSION MUST DENOTE AN INTEGER IN THE VARIABLE'S RANGE. IF THE RANGE OF THE VARIABLE IS \texttt{I1,I2,...,IN} AND THE VALUE EXPRESSION IS \texttt{X}, THEN THE CONSTRAINTS ADDED TO THE CONTEXT ARE \texttt{\texttt{V}I1-1<XV} AND \texttt{\texttt{V}X<IN+1V}. 

3. CONSTRAINT PROCESSING

When a constraint is added to a context in ARF it goes through three phases of processing. In Phase 1 the constraint is simplified. In Phase 2 tests are made to determine if the constraint implies that a new value can be set in the context or that elements can be deleted from a variable's range. In Phase 3 further deductions are attempted by considering the new constraint in conjunction with the other constraints in the context. In this section we will describe the processing that occurs in each of these phases.

3.1. Phase 1

Phase 1 evaluates a constraint by calling the expression evaluation executive routine described earlier. If the constraint reduces to true, then Phase 1 exits indicating success. If it reduces to false, then Phase 1 exits indicating failure. In all other cases Phase 2 is called to continue the processing and determine the exit conditions.

3.2. Phase 2

Phase 2 further examines a constraint in an attempt to make
DEDUCTIONS FROM IT. THERE IS AN ACTION ROUTINE CORRESPONDING TO EACH MAIN OPERATOR A CONSTRAINT MAY HAVE (I.E. =, <, ~, \textsc{vexclv}, ^, AND \textsc{v}), AND THE PHASE 2 EXECUTIVE SIMPLY DETERMINES A CONSTRAINT'S MAIN OPERATOR AND CALLS THE APPROPRIATE ACTION ROUTINE.

EACH OF THE PHASE 2 ACTIONS CURRENTLY DEFINED IN ARF BEGINS BY CALLING A SUBROUTINE WHICH TESTS IF THE CONSTRAINT CONTAINS OCCURRENCES OF EXACTLY ONE VARIABLE (WE WILL REFER TO THIS SUBROUTINE AS R105, ITS INTERNAL NAME IN THE PROGRAM). IF THAT IS THE CASE, THEN THE SUBROUTINE GENERATES THE RANGE VALUES OF THAT VARIABLE AND EVALUATES THE CONSTRAINT FOR EACH GENERATED VALUE. WHEN THE CONSTRAINT IS FALSE FOR A VALUE, THAT VALUE IS DELETED FROM THE VARIABLE'S RANGE. AT COMPLETION OF THE GENERATION SEVERAL DEDUCTIONS ARE POSSIBLE. IF THE CONSTRAINT REDUCED TO TRUE OR FALSE FOR EACH RANGE VALUE, THEN IT MAY BE DISCARDED. (NOTE, THIS NEED NOT BE THE CASE. THE CONSTRAINT $\textsc{v}b[\text{s}(2)] = 3\textsc{v}$ IN A CONTEXT WHOSE DATA STRUCTURE HAS NO VALUES IN THE B VECTOR CANNOT BE DISCARDED.) IF ALL THE VALUES HAVE BEEN DELETED FROM THE RANGE, THEN THE CONSTRAINT IS FALSE IN THE CONTEXT AND A FAILURE EXIT MAY BE TAKEN. IF ALL BUT ONE OF THE RANGE ELEMENTS HAVE BEEN DELETED, THEN THE REMAINING RANGE ELEMENT MAY BE ASSIGNED AS THE VALUE OF THE VARIABLE.

TO ILLUSTRATE THE EFFECT OF THIS SUBROUTINE WE CAN REFER
Again to the first subproblem of the crypt-addition problem shown in Figure III.1. If this subroutine processed the constraint \( \forall S(7) = 0 \), it would evaluate the constraint with each of the values in the range of \( S(7) \). The constraint would be false when the value of \( S(7) \) is 0 and true for all other values of \( S(7) \). Hence, 0 would be deleted from the range of \( S(7) \) and the constraint would be discarded. If the subroutine were then given \( \forall S(7) = 0 \) to process, it would find the constraint false for all values in the range of \( S(7) \) and indicate a failure exit. Similarly, if \( \forall S(7) = 0 \) were given to the subroutine before \( \forall S(7) = 0 \), then it would set 0 as the value of \( S(7) \). This would allow the constraint \( \forall S(4) + S(7) < 10 \) to be reduced to \( \forall S(4) < 10 \). If \( \forall S(4) < 10 \) were then given to the subroutine it would be true for all values in the range of \( S(4) \) and therefore could be discarded.

Routine R105 is simple and effective, but it is expensive to apply for variables which have large ranges. An alternative way of making these deductions which does not depend on range size is to test for particular constraint forms such as \( \forall S(i) = j \), \( \forall S(i) = j \), \( \forall S(i) < j \), and \( j < S(i) \), where \( j \) is an \( \forall \text{INTEV} \) expression. An action routine can be written for each form which acts directly on a range list to make the indicated changes. A routine organized in this way would in most cases require less processor time to make
ITS DEDUCTIONS THAN DOES R105, BUT A LARGE NUMBER OF ACTION ROUTINES ARE REQUIRED TO ACHIEVE THE GENERALITY OF THE ARP ROUTINE. IF THE REF LANGUAGE WERE EXTENDED TO ALLOW SPECIFICATION OF INFINITE RANGES, THEN A ROUTINE WHICH LOOKED FOR CONSTRAINT FORMS WOULD BE NECESSARY SINCE GENERATION OF EACH RANGE ELEMENT WOULD NOT BE POSSIBLE.

WE WILL NOW DESCRIBE THE PHASE 2 ACTION ROUTINES FOR EACH CONSTRAINT MAIN OPERATOR. THE ACTIONS FOR <, -, AND v DO NO MORE THAN CALL ROUTINE R105. IF ROUTINE R105 DISCARDS THE CONSTRAINT OR INDICATES failure, THEN THE ACTIONS EXIT APPROPRIATELY. OTHERWISE, CONTROL IS PASSED TO THE PHASE 3 PROCESSOR TO INCORPORATE THE CONSTRAINT INTO THE CONTEXT.

THE ACTION FOR EQUATIONS ATTEMPTS TO FORM A VALUE EXPRESSION FROM ANY EQUATION WHICH IS NOT DISCARDED BY R105. THE VALUE EXPRESSION IS FORMED BY SEARCHING THE CONSTRAINT FOR A SUBEXPRESSION X WHICH SATISFIES EACH OF THE FOLLOWING:

1. THE CONSTRAINT IS IN ONE OF THE FOLLOWING FORMS:
   \[ x = y, \quad y = x, \quad -x = y, \quad y = -x, \quad \ldots + x + \ldots = y, \quad y = \ldots + x + \ldots, \]
   \[ \ldots + - x + \ldots = y, \quad \text{or} \quad y = \ldots + - x + \ldots \] WHERE Y IS ANY EXPRESSION.

2. THE EXPRESSION X IS IN ONE OF THE FOLLOWING
FORMS: S(I), Y[J] WHERE Y IS A SYMBOL EXPRESSION AND J IS AN INTEGER EXPRESSION, OR Y OF Z WHERE Y AND Z ARE SYMBOL EXPRESSIONS.

3. THE EXPRESSION X OCCURS ONLY ONCE IN THE CONSTRAINT.


THE ACTION FOR CONJUNCTIONS MAKES A SEPARATE CONSTRAINT OUT OF EACH CONJUNCT AND ADDS THE CONSTRAINTS TO THE CONTEXT VIA RECURSIVE CALLS OF THE PHASE 2 PROCESSOR.

THE ACTION FOR VEXCLv CONSTRAINTS MAKES MULTIPLE CALLS ON ROUTINE R105 TAKING ADVANTAGE OF THE FACT THAT ANY SUBSET OF AN VEXCLv CONSTRAINT'S OPERANDS MUST BE DISTINCT. THAT IS, THE ACTION GROUPS TOGETHER OPERANDS WHICH ARE CONSTANTS AND OPERANDS WHICH CONTAIN OCCURRENCES OF THE SAME VARIABLE (AND NO OTHER VARIABLE) TO FORM NEW VEXCLv CONSTRAINTS. THESE CONSTRAINTS ARE THEN INPUT TO ROUTINE R105. SUCH CALLS TO R105 CANNOT CAUSE THE ORIGINAL CONSTRAINT TO BE DISCARDED, BUT ALL OTHER DEDUCTIONS MADE BY ROUTINE R105 ARE VALID. FOR EXAMPLE, CONSIDER THE CONSTRAINT VEXCL(1,S(2),S(2)+1,S(3))v. THE ACTION FIRST MAKES A CALL ON R105 WITH THE ENTIRE CONSTRAINT WHICH DOES NOT RESULT IN ANY DEDUCTIONS. THE ACTION THEN FORMS THE CONSTRAINT VEXCL(1,S(2),S(2)+1)v AND INPUTS IT TO R105. R105 CAN ELIMINATE 0 AND 1 FROM THE RANGE OF S(2), BUT ITS INDICATION THAT THE CONSTRAINT BE DISCARDED IS IGNORED. THE ACTION THEN FORMS VEXCL(1,S(3))v AND INPUTS IT TO R105. THIS RESULTS IN THE DELETION OF 1 FROM THE RANGE OF S(3). THE ACTION WILL THEN CALL THE PHASE PROCESSOR TO INCORPORATE THE ORIGINAL CONSTRAINT INTO THE CONTEXT.
3.3. PHASE 3

Any constraint which is not discarded during phase 2 is input to the phase 3 processor. Phase 3 attempts to make deductions by considering the new constraint in conjunction with the other constraints in the context. It is also organized as an executive routine and a set of action routines. The actions are stored in lists indexed by operator pairs. The executive pairs a new constraint with each existing constraint in the context and executes the list of actions associated with the operator pair formed by the main operators of the two constraints. For example, there is a list of actions indexed by the pair \( (=,\lor) \); if an equation is being added to a context which contains a disjunctive constraint, then when the executive pairs the equation with the disjunction the actions on the list indexed by the pair \( (=,\lor) \) will be executed. The action routines and index that are in the current AFP are shown in Appendix II. These actions are designed to detect obvious inconsistencies (e.g., \(-(X) \) and \( X \)), delete redundant constraints (e.g., \( X \) and \( \lor (...)\)), and provide further simplification (e.g., given \( =(X,\text{INTE}(I)) \), replace \( X \) by \( \text{INTE}(I) \) in the other constraints).

After a new constraint has been paired with each existing constraint by the phase 3 executive, it is added to the context.
LIST OF CONSTRAINTS. ANY NEW CONSTRAINTS FORMED DURING THE APPLICATION OF THE ACTIONS ARE ADDED TO THE CONTEXT VIA A RECURSIVE CALL OF PHASE 1.

4. EXAMPLE

TO ILLUSTRATE THE BEHAVIOR OF ARFS CONSTRAINT AND EXPRESSION MANIPULATION ROUTINES CONSIDER AGAIN THE CRYPT-ADDITION SUBPROBLEM IN WHICH THE CARRY VALUE IS ALWAYS ZERO. AS THE INTERPRETER MOVES THROUGH THE L2 LOOP THE FIRST TIME THE CONTEXT WHICH CONTAINS THIS SUBPROBLEM HAS THE FOLLOWING FORM AFTER BRANCHING TO STATEMENT L3:

CONTEXT

DATA STRUCTURE
CARRY
  <CARRY>: 0
Y
  <Y>: S(8)
E
  <E>: S(3)
D
  <D>: S(1)
A1
  VECTOR: X, S, E, N, D
A2
  VECTOR: X, M, O, P, E
SUM
  VECTOR: M, O, N, E, Y
I
  <I>: 5
.. 
.. 
VARIABLES
S(8)
RANGE: 0 1 2 3 4 5 6 7 8 9
S(7) RANGE: 1 2 3 4 5 6 7 8 9
S(4) RANGE: 1 2 3 4 5 6 7 8 9
S(3) RANGE: 0 1 2 3 4 5 6 7 8 9
S(1) RANGE: 0 1 2 3 4 5 6 7 8 9

CONSTRAINTS
S(1)+S(3)<10
EXCL(S(1),S(2),...,S(8))

Note that the constraint \(-(M>0)^-<(S>0)^-\) from the
\texttt{vcondition} statement of line 9 does not appear in the context.
It is discarded while being processed at the time it is added to
the context as follows. In phase 1 the constraint is evaluated and
becomes \(-S(7)=0\land-S(4)=0\). Phase 2 divides the conjunction
into two constraints and recurses to process them. When routine
R105 evaluates the constraints for each value in the ranges of
S(4) and S(7) during the recursive call of phase 2, the
constraints are discarded and zero is deleted from the range of
both variables.

At the L3 \texttt{vcondition} statement the interpreter presents to
the constraint manipulation methods the expression \(<A1[^<I>]>+\n<A2[^<I>]>+<\text{carry}>=<\text{sum[^<I>]>}\) as a new constraint. The phase 1
processor calls the expression evaluation executive which reduces
THE EXPRESSION TO $\forall S(1)+S(3)-S(8)=0$. THE EXPRESSION IS INPUT TO
THE PHASE 2 PROCESSOR WHICH CALLS ITS ACTION ROUTINE FOR
EQUATIONS. THIS ACTION ROUTINE CALLS THE VALUE SETTING ROUTINE TO
SET THE VALUE OF $S(1)$ TO BE $\forall -S(3)+S(8)$. THE VALUE SETTING
ROUTINE INSERTS THE VALUE IN THE CONTEXT'S VARIABLE STRUCTURE,
REPLACES $S(1)$ BY ITS VALUE IN THE CONTEXT'S TWO CONSTRAINTS, AND
CREASES THE NEW CONSTRAINTS $\forall -1<-S(3)+S(8)$ AND $\forall -S(3)+S(8)<10$. 
THESE FOUR NEW CONSTRAINTS ARE INPUT TO THE PHASE 1 PROCESSOR FOR
INCORPORATION INTO THE CONTEXT. WHEN THE VALUE OF $S(1)$ WAS
SUBSTITUTED IN $\forall S(1)+S(3)<10$, IT BECAME $\forall -S(3)+S(8)+S(3)<10$. IN
PHASE 1 THIS CONSTRAINT REDUCES TO $\forall S(8)<10$. IN PHASE 2 THE
CONSTRAINT IS EVALUATED FOR EACH VALUE IN THE RANGE OF $S(8)$ AND
FOUND TO BE IRRELEVANT. WHEN ALL THE CONSTRAINTS HAVE BEEN
PROCESSED AND INTERPRETATION PROCEEDS THE CONTEXT HAS THE
FOLLOWING FORM:

CONTEXT

DATA STRUCTURE


VARIABLES

$S(1)$

VALUE: $-S(3)+S(8)$

RANGE: 0 1 2 3 4 5 6 7 8 9


CONSTRAINTS

EXCL($-S(3)+S(8), S(2), S(3), \ldots, S(8)$)

$-S(3)+S(8)<10$

$-1<-S(3)+S(8)$
V. SOLVING THE SEARCH PROBLEM IN THE SUBPROBLEM SPACE

In Chapter III we observed that ARF is organized to treat a REF procedure as a heuristic search problem in a space whose objects are the context structures used by the ARF interpreter and whose operators are the statements in the REF procedure. The search begins by applying the operator associated with the \texttt{vbegin} statement to an empty context and ends successfully when the operator associated with the \texttt{vend} statement can be applied to produce an object with no unsatisfied constraints. A problem solver for such a problem must have facilities for selecting an object from among those at the terminal nodes of the search tree, for selecting an operator which is applicable to the selected object, for applying the operator to the object, and for testing if an object represents a solution to the problem. In this chapter we will discuss the characteristics of this search problem and consider ways of providing a program with the necessary facilities for solving it.

A. OPERATORS AND THEIR APPLICATION

In this section we wish to describe in detail the operators used by ARF in the subproblem space and explore other operator formulations which might be used to improve the effectiveness of
THE PROGRAM.

1. ARFVS OPERATOR FORMULATION

ARFVS INPUT ROUTINE TRANSLATES A REF PROCEDURE INTO A DIRECTED GRAPH. VERTICES OF THE GRAPH ARE FORMED BY LABELED STATEMENTS, IFV STATEMENTS, COMPUTED GOTOV STATEMENTS, BEGINV, AND ENDV. PATHS CONNECTING THE VERTICES REPRESENT SEQUENCES OF VSETV, VSet.VECTORV, AND VCONDITIONV STATEMENTS. STANDARD GOTOV STATEMENTS ARE IMPLICITLY REPRESENTED IN THE TOPOLOGICAL STRUCTURE OF THE GRAPH. FORV LOOPS ARE TRANSLATED INTO VSETV, IFV, AND GOTOV STATEMENTS AS SHOWN IN FIGURE V.1. FIGURE V.2 ILLUSTRATES THIS REPRESENTATION OF A REF PROCEDURE BY SHOWING THE DIRECTED GRAPH FOR THE REF STATEMENT OF THE MONKEY PROBLEM DISCUSSED IN CHAPTER II.
GIVEN THE VFORV LOOP

FOR I = <N> DO TO L1;

L1:

ARFVS INPUT ROUTINE WILL TRANSLATE IT INTO THE FOLLOWING STATEMENTS:

SET <I> TO 1;
$2$: IF <I> = <N> + 1 THEN $1;

L1:
SET <I> TO <I> + 1;
GOTO $2 ;

$1:

WHERE $1$ AND $2$ ARE INTERNALLY GENERATED LABELS.

FIGURE V.1. EXAMPLE OF ARFVS VFORV LOOP TRANSLATION.
vertices
A. BEGIN
B. WALK
C. if ¬(MONKEY[1] = BOX[1]) then WALK;
D. LI
E. goto <M> (WALK, CLIMB, MOVE.BOX);
F. CLIMB
G. if ¬(MONKEY[1] = UNDER.BANANAS) then STEP.DOWN;
H. goto <M> (GET.BANANAS, STEP.DOWN);
I. END
J. STEP.DOWN
K. MOVE.BOX

paths
a. set vector X to X1, X2, UNDER.BANANAS;
   b. set MONKEY[1] to X[select(1,3)];
   c. empty
   d. empty
   e. set <M> to select(1,3);
   f. empty
   g. empty
   h. empty
   i. set MONKEY[2] to ON.BOX;
   j. set <M> to select(1,2);
   k. empty
   l. empty
   m. empty
   n. set MONKEY[2] to ON.FLOOR;
   o. set MONKEY[1] to X[select(1,3)];
   p. empty
   q. empty
   r. empty
   s. empty

Figure V.2. Directed graph for the monkey problem
ARFSV OPERATORS IN THE SUBPROBLEM SPACE ARE FORMULATED IN TERMS OF THE DIRECTED GRAPH IN THAT THE SEQUENCE OF STATEMENTS REPRESENTED BY A PATH IN THE GRAPH AND THE PATH'S TERMINAL VERTEX ARE TREATED AS A SINGLE OPERATOR. THE SET OF OPERATORS WHICH MAY BE DEFINED IN THIS WAY ARE DESCRIBED IN FIGURE V.3. CONSIDER THE OPERATORS WHICH WOULD BE DEFINED BY THE REF STATEMENT OF THE MONKEY PROBLEM SHOWN IN FIGURE II.9. THE FIRST OPERATOR FOR THIS PROCEDURE WOULD INCLUDE THE $\texttt{BEGIN}$ STATEMENT AND THE FIRST FOUR $\texttt{SET}.\texttt{VECTOR}$ STATEMENTS. APPLICATION OF THIS OPERATOR TO A CONTEXT WOULD CAUSE INTERPRETATION OF THE FOUR $\texttt{SET}.\texttt{VECTOR}$ STATEMENTS AND ATTACHMENT OF THE OPERATOR WHICH BEGINS AT STATEMENT $\texttt{WALK}$ TO EACH OF THE RESULTING CONTEXTS. THE OPERATOR WHICH BEGINS AT STATEMENT $\texttt{WALK}$ INCLUDES ONLY THE $\texttt{SET}$ STATEMENT AT $\texttt{WALK}$ AND THE $\texttt{IF}$ STATEMENT AT LINE 7 OF THE PROCEDURE. APPLICATION OF THIS OPERATOR WOULD CAUSE INTERPRETATION OF STATEMENT $\texttt{WALK}$ AND THE FORMATION OF NEW CONTEXTS AS INDICATED BY THE $\texttt{IF}$ STATEMENT. EACH OF THE OPERATORS DEFINED BY THIS PROCEDURE ARE DESCRIBED IN FIGURE V.4.
OPERATORS DEFINED IN THE DIRECTED GRAPH

OPERATORS

(OPERATORS INPUT ONE OBJECT AND MAY PRODUCE MULTIPLE OUTPUT OBJECTS.)

OPERATOR FOR ^BEGIN^.
TRANSFER TO THE ACTION FOR THE STATEMENT AT LINE 2 OF THE PROCEDURE WITH THE INPUT OBJECT AS INPUT.

OPERATOR FOR A ^SET^ STATEMENT AT LINE K OF A PROCEDURE.
TRANSFER TO THE ACTION FOR AN UNLABELED ^SET^ STATEMENT AT LINE K OF A PROCEDURE WITH THE INPUT OBJECT AS INPUT.

OPERATOR FOR A ^SET^ VECTOR^ STATEMENT AT LINE K OF A PROCEDURE.
TRANSFER TO THE ACTION FOR AN UNLABELED ^SET^ VECTOR^ STATEMENT AT LINE K OF A PROCEDURE WITH THE INPUT OBJECT AS INPUT.

OPERATOR FOR A ^CONDITION^ STATEMENT AT LINE K OF A PROCEDURE.
TRANSFER TO THE ACTION FOR AN UNLABELED ^CONDITION^ STATEMENT AT LINE K OF A PROCEDURE WITH THE INPUT OBJECT AS INPUT.

OPERATOR FOR STATEMENT ^GOTO X^.
TRANSFER TO THE ACTION FOR UNLABELED STATEMENT ^GOTO X^ WITH THE INPUT OBJECT AS INPUT.

OPERATOR FOR STATEMENT ^GOTO X (Y_1,Y_2,...,Y_N)^.
TRANSFER TO THE ACTION FOR UNLABELED STATEMENT ^GOTO X (Y_1,Y_2,...,Y_N)^ WITH THE INPUT OBJECT AS INPUT.

OPERATOR FOR AN ^IF^ STATEMENT AT LINE K OF A PROCEDURE.
TRANSFER TO THE ACTION FOR AN UNLABELED ^IF^ STATEMENT AT LINE K OF A PROCEDURE WITH THE INPUT OBJECT AS INPUT.

ACTIONS

(ACTIONS INPUT AND OUTPUT MULTIPLE OBJECTS.)

ACTION FOR AN UNLABELED ^SET^ STATEMENT AT LINE K OF A PROCEDURE.
FOR EACH INPUT OBJECT CALL THE ^SET^ STATEMENT INTERPRETER

FIGURE V.3. OPERATORS DEFINED IN THE DIRECTED GRAPH
WITH THE STATEMENT AND THE OBJECT AS INPUT. TRANSFER TO THE ACTION FOR THE STATEMENT AT LINE K+1 OF THE PROCEDURE WITH THE OBJECTS OUTPUT BY THE \vsset\ statement interpreter AS INPUT.

**ACTION FOR UNLABELED STATEMENT** \vsset\ VECTOR X TO Y1,Y2,...,YN AT LINE K OF A PROCEDURE.

FOR EACH INPUT OBJECT CALL THE \vsset\ STATEMENT INTERPRETER WITH THE STATEMENT \vsset x[1] TO Y1 AND THE OBJECT. FOR I=2,3,...,N AND FOR EACH OF THE OBJECTS PRODUCED DURING THE INTERPRETATION OF THE X[I-1] \vsset\ STATEMENT, CALL THE \vsset\ STATEMENT INTERPRETER WITH THE STATEMENT \vsset x[I] TO YI AND THE OBJECT. TRANSFER TO THE ACTION FOR THE STATEMENT AT LINE K+1 OF THE PROCEDURE WITH THE RESULTING OBJECTS.

**OPERATOR FOR AN UNLABELED \vcondition\ STATEMENT AT LINE K OF A PROCEDURE.**

FOR EACH INPUT OBJECT ADD THE BOOLEAN EXPRESSION OF THE STATEMENT AS A CONSTRAINT TO THE OBJECT. TRANSFER TO THE ACTION FOR THE STATEMENT AT LINE K+1 OF THE PROCEDURE WITH THE OBJECTS OUTPUT BY THE \v\v\add\ constraint \v\v routine AS INPUT.

**ACTION FOR UNLABELED STATEMENT** \vgoto X\.

SET AS THE NEXT APPLICABLE OPERATOR IN EACH OF THE INPUT OBJECTS THE OPERATOR FOR STATEMENT X. OUTPUT THE INPUT OBJECTS.

**ACTION FOR UNLABELED STATEMENT** \vgoto X (Y1,Y2,...,YN)\.

FOR EACH INPUT OBJECT AND FOR I=1,2,...,N-1 SET \vx=I AS A CONSTRAINT IN A COPY OF THE OBJECT AND SET THE OPERATOR FOR STATEMENT YI AS THE NEXT APPLICABLE OPERATOR IN THE RESULTING OBJECT. IN ADDITION, FOR EACH INPUT OBJECT ADD \vx=N AS A CONSTRAINT TO THE OBJECT AND SET THE OPERATOR FOR STATEMENT YN AS THE NEXT APPLICABLE OPERATOR IN THE RESULTING OBJECT. OUTPUT ALL RESULTING OBJECTS.

**ACTION FOR UNLABELED STATEMENT** \vif X THEN Y AT LINE K OF A PROCEDURE.

FOR EACH INPUT OBJECT ADD X AS A CONSTRAINT TO A COPY OF THE OBJECT AND SET THE OPERATOR FOR STATEMENT Y AS THE NEXT APPLICABLE OPERATOR IN THE RESULTING OBJECT. IN ADDITION, FOR EACH INPUT OBJECT ADD \(\neg(X)\) AS A CONSTRAINT TO THE OBJECT AND

FIGURE V.3 (CONTINUED)
SET THE OPERATOR FOR THE STATEMENT AT LINE K+1 AS THE NEXT APPLICABLE OPERATOR IN THE RESULTING OBJECT. OUTPUT ALL RESULTING OBJECTS.

ACTION FOR A LABELED VSET\textsuperscript{\texttt{V}} STATEMENT AT LINE K OF A PROCEDURE.
SET AS THE NEXT APPLICABLE OPERATOR IN EACH OF THE INPUT OBJECTS THE OPERATOR FOR A VSET\textsuperscript{\texttt{V}} STATEMENT AT LINE K OF THE PROCEDURE. OUTPUT THE INPUT OBJECTS.

ACTION FOR A LABELED VSET\textsuperscript{\texttt{VECTOR}} STATEMENT AT LINE K OF A PROCEDURE.
SET AS THE NEXT APPLICABLE OPERATOR IN EACH OF THE INPUT OBJECTS THE OPERATOR FOR A VSET\textsuperscript{\texttt{VECTOR}} STATEMENT AT LINE K OF THE PROCEDURE. OUTPUT THE INPUT OBJECTS.

ACTION FOR A LABELED VCONDITION\textsuperscript{\texttt{V}} STATEMENT AT LINE K OF A PROCEDURE.
SET AS THE NEXT APPLICABLE OPERATOR IN EACH OF THE INPUT OBJECTS THE OPERATOR FOR A VCONDITION\textsuperscript{\texttt{V}} STATEMENT AT LINE K OF A PROCEDURE. OUTPUT THE INPUT OBJECTS.

ACTION FOR LABELED STATEMENT V\texttt{GOTO X}.
SET AS THE NEXT APPLICABLE OPERATOR IN EACH OF THE INPUT OBJECTS THE OPERATOR FOR STATEMENT V\texttt{GOTO X}. OUTPUT THE INPUT OBJECTS.

ACTION FOR STATEMENT V\texttt{GOTO X (Y1,Y2,...,YN)}.
SET AS THE NEXT APPLICABLE OPERATOR IN EACH OF THE INPUT OBJECTS THE OPERATOR FOR STATEMENT V\texttt{GOTO X (Y1,Y2,...,YN)}. OUTPUT THE INPUT OBJECTS.

ACTION FOR A LABELED V\texttt{IF} STATEMENT AT LINE K OF A PROCEDURE.
SET AS THE NEXT APPLICABLE OPERATOR IN EACH OF THE INPUT OBJECTS THE OPERATOR FOR AN V\texttt{IF} STATEMENT AT LINE K OF A PROCEDURE. OUTPUT THE INPUT OBJECTS.

ACTION FOR LABELED OR UNLABELED V\texttt{END}.
FOR EACH INPUT OBJECT CONDUCT THE BACKTRACKING SEARCH TO FIND A SET OF VARIABLE VALUES WHICH SATISFY THE CONSTRAINTS IN THE OBJECT; IF THE SEARCH IS SUCCESSFUL, THEN EXIT THE ACTION WITH AN OUTPUT OBJECT WHICH CONTAINS THE SUCCEEDING SET OF VARIABLE VALUES. IF ALL THE SEARCHES FAIL, THEN EXIT WITH NO OUTPUT OBJECTS.

FIGURE V.3 (CONTINUED).
OPERATOR FOR \texttt{BEGIN}^\texttt{v} \\
\texttt{EXECUTE ACTION FOR UNLABELED STATEMENT \texttt{SET VECTOR} X TO} \\
\texttt{X1,X2,UNDER.BANANASv. EXECUTE ACTION FOR UNLABELED STATEMENT} \\
\texttt{\texttt{SET VECTOR} Y TO ON.FLOOR,ON.BOXv. EXECUTE ACTION FOR} \\
\texttt{UNLABELED STATEMENT \texttt{SET VECTOR} MONKEY TO X1,ON.FLOORv.} \\
\texttt{EXECUTE ACTION FOR UNLABELED STATEMENT \texttt{SETVECTOR} BOX TO} \\
\texttt{X2,ON.FLOORv. EXECUTE ACTION FOR LABELED STATEMENT \texttt{WALK} :} \\
\texttt{SET MONKEY[1] TO X[SELECT(1,3)]v.} \\

OPERATOR FOR STATEMENT \texttt{WALK}^\texttt{v} \\
\texttt{EXECUTE ACTION FOR UNLABELED STATEMENT \texttt{SET MONKEY[1] TO}} \\
\texttt{X[SELECT(1,3)]v. EXECUTE ACTION FOR UNLABELED STATEMENT \texttt{IF}} \\
\texttt{\texttt{IF} \neg (\texttt{MONKEY[1]} = \texttt{BOX[1]}) \texttt{THEN WALK}^\texttt{v.}} \\

OPERATOR FOR STATEMENT L1 \\
\texttt{EXECUTE ACTION FOR UNLABELED STATEMENT \texttt{SET} \texttt{M} \texttt{TO}} \\
\texttt{SELECT(1,3)v. EXECUTE ACTION FOR UNLABELED STATEMENT \texttt{GOTO}} \\
\texttt{\texttt{GOTO} \texttt{M} (\texttt{WALK,CLIMB,MOVE.BOX})v.} \\

OPERATOR FOR STATEMENT \texttt{CLIMB}^\texttt{v} \\
\texttt{EXECUTE ACTION FOR UNLABELED STATEMENT \texttt{SET MONKEY[2] TO}} \\
\texttt{ON.BOXv. EXECUTE ACTION FOR UNLABELED STATEMENT \texttt{IF}} \\
\texttt{\texttt{IF} \neg (\texttt{MONKEY[1]} = \texttt{UNDER.BANANAS}) \texttt{THEN STEP.DOWN}^\texttt{v.}} \\

OPERATOR FOR STATEMENT \texttt{SET} \texttt{M} \texttt{TO} \\
\texttt{SELECT(1,2)v} \\
\texttt{EXECUTE ACTION FOR UNLABELED STATEMENT \texttt{SET} \texttt{M} \texttt{TO}} \\
\texttt{SELECT(1,2)v. EXECUTE ACTION FOR UNLABELED STATEMENT \texttt{GOTO}} \\
\texttt{\texttt{GOTO} \texttt{M} (\texttt{GET.BANANAS,STEP.DOWN})v.} \\

OPERATOR FOR STATEMENT \texttt{STEP.DOWN}^\texttt{v} \\
\texttt{EXECUTE ACTION FOR UNLABELED STATEMENT \texttt{SET MONKEY[2] TO}} \\
\texttt{ON.FLOORv. EXECUTE ACTION FOR UNLABELED STATEMENT \texttt{GOTO} L1v.} \\

OPERATOR FOR STATEMENT \texttt{MOVE.BOX}^\texttt{v} \\
\texttt{EXECUTE ACTION FOR UNLABELED STATEMENT \texttt{SET MONKEY[1] TO}} \\
\texttt{X[SELECT(1,3)]v. EXECUTE ACTION FOR UNLABELED STATEMENT \texttt{SET}} \\
\texttt{BOX[1] TO MONKEY[1]v. EXECUTE ACTION FOR UNLABELED STATEMENT} \\
\texttt{\texttt{GOTO} L1v.} \\

OPERATOR FOR STATEMENT \texttt{GET.BANANAS}^\texttt{v} \\
\texttt{EXECUTE THE ACTION FOR AN \texttt{VEND}v STATEMENT.} \\

\textbf{FIGURE V.4. ARFS OPERATORS FOR THE MONKEY PROBLEM}
AN OPERATOR IS APPLIED IN ARF BY CALLING THE INTERPRETER WITH A SINGLE CONTEXT AS INPUT. THE INPUT CONTEXT CONTAINS THE NAME OF A PATH AND THE INTERPRETER PERFORMS THE ACTIONS DEFINED FOR THE STATEMENTS ON THAT PATH AND AT ITS TERMINAL VERTEX. THE INTERPRETER EXITS EITHER WITH A LIST OF OUTPUT CONTEXTS OR WITH A SOLUTION CONTEXT PLUS A SIGNAL INDICATING THAT A SOLUTION WAS FOUND.

NOTE THAT IN THIS FORMULATION OF THE SEARCH PROBLEM THERE IS NEVER MORE THAN ONE OPERATOR APPLICABLE TO AN OBJECT AT ANY GIVEN TIME. THE SEARCH TREE GROWS EXponentially because operators WHICH INPUT A SINGLE OBJECT CAN PRODUCE MULTIPLE OUTPUT OBJECTS. IN THE STANDARD FORMULATION OF A SEARCH PROBLEM TYPICALLY USED FOR GAME PLAYING AND THEOREM PROVING PROGRAMS, EACH OPERATOR HAS ONE OUTPUT AND THE TREE GROWS EXponentially because more THAN ONE OPERATOR IS APPLICABLE TO AN OBJECT AT ANY GIVEN TIME. CONSIDER THE SIMILARITIES AND DIFFERENCES BETWEEN THESE TWO FORMULATIONS.

THE SEARCH FORMULATION FOR PLAYING CHESS INTRODUCED BY CLAUDE SHANNON (1950) AND UTILIZED BY A. M. TURING (1953) AND BY A GROUP AT LOS ALAMOS (KISTER ET AL., 1957) CONTAINS A SINGLE OPERATOR AS FOLLOWS:

FIND AND DENOTE By \( M_1, M_2, \ldots, M_K \) THE LEGAL MOVES FOR THE INPUT BOARD POSITION. FOR \( I = 1, 2, \ldots, K-1 \) FORM AN OUTPUT OBJECT BY MAKING
MOVE MI IN A COPY OF THE INPUT OBJECT. FORM AN OUTPUT OBJECT BY MAKING MOVE MK IN THE INPUT OBJECT.

THIS OPERATOR TAKES A SINGLE BOARD POSITION AS INPUT AND PRODUCES MULTIPLE BOARD POSITIONS AS OUTPUT BY MAKING ALL POSSIBLE LEGAL MOVES IN THE INPUT BOARD POSITION. SINCE THIS IS THE ONLY OPERATOR IN THE SEARCH SPACE, THE FORMULATION HAS THE SAME \texttt{ONE APPLICABLE OPERATOR} PROPERTY THAT WE OBSERVED FOR ARFS FORMULATION.

WHEN THERE IS ONLY ONE OPERATOR APPLICABLE TO EACH OBJECT THE PROBLEM SOLVER NEED ONLY CONCERN ITSELF WITH SELECTING OBJECTS DURING THE SEARCH RATHER THAN SELECTING BOTH OBJECTS AND OPERATORS. IT IS AS IF IN THE STANDARD HEURISTIC SEARCH FORMULATION ONCE AN OBJECT IS SELECTED, ALL THE OPERATORS APPLICABLE TO THAT OBJECT ARE APPLIED TO IT. THE ADVANTAGES GAINED FROM THIS SIMPLIFICATION OF THE PROBLEM SOLVER'S SELECTION MECHANISMS MUST BE WEIGHTED AGAINST THE PROCESSING TIME AND MEMORY REQUIRED TO PRODUCE ALL THE OBJECTS POSSIBLE FROM A SELECTED OBJECT. THAT IS, GIVEN THE STANDARD FORMULATION AN OPERATOR SELECTION MECHANISM CAN BE USED TO ELIMINATE THE NEED OF APPLYING ALL THE OPERATORS TO THE SELECTED OBJECT.

ONE WAY OF ALLEVIATING THE PROBLEM OF PRODUCING IRRELEVANT OBJECTS IN THE ONE APPLICABLE OPERATOR FORMULATION IS TO CONSTRUCT THE OPERATORS WHICH PRODUCE MORE THAN ONE OUTPUT OBJECT AS
GENERATORS. SUCH AN OPERATOR WOULD STOP AFTER PRODUCING AN OUTPUT OBJECT AND COULD BE RESTARTED AT ANY GIVEN TIME TO PRODUCE ITS NEXT OUTPUT OBJECT. GIVEN SUCH GENERATORS THE SEARCH EXECUTIVE COULD APPLY ITS OBJECT SELECTION ROUTINE AFTER THE PRODUCTION OF EACH NEW OBJECT. IF A NEW OBJECT IS SELECTED AND IT EVENTUALLY LEADS TO A SOLUTION, THEN THE GENERATOR NEED NOT COMPLETE ITS PRODUCTION OF OUTPUT OBJECTS. THE NEWELL, SHAW, AND SIMON CHESS PROGRAM (NEWELL ET AL., 1958) PROVIDES AN EXAMPLE OF THE USE OF GENERATORS IN THIS WAY. THE PROGRAM CONTAINS A SET OF MOVE GENERATORS AND EACH GENERATED MOVE IS EVALUATED BEFORE THE NEXT ONE IS GENERATED. IF THE EVALUATION ROUTINES FIND THAT THE MOVE EXCEEDS A GIVEN ACCEPTABILITY THRESHOLD, THEN THE PROGRAM MAKES THE ACCEPTABLE MOVE WITHOUT RESTARTING THE GENERATORS TO CONSIDER THE REMAINING LEGAL MOVES.

ONE CAN FURTHER IMPROVE THE EFFECTIVENESS OF OPERATORS CONSTRUCTED AS GENERATORS BY PROVIDING THEM WITH THE CAPABILITY OF ALTERING THE ORDER IN WHICH THEY GENERATE THE OUTPUT OBJECTS. FOR EXAMPLE, IN THE CHESS PROGRAM OF EERNSTEIN (BERNSTEIN ET AL., 1958) THE OPERATOR (I.E., THE LEGAL MOVE GENERATOR) GENERATES FIRST THOSE MOVES WHICH INVOLVE KING SAFETY, THEN THOSE WHICH INVOLVE DEFENDING ONE'S OWN MEN, THEN THOSE WHICH INVOLVE ATTACKING THE OPPONENT'S MEN, ETC. AS ANOTHER EXAMPLE, IN ARF'S SUBPROBLEM SPACE ONE MIGHT WISH TO REQUEST THAT THE ACTION FOR A
COMPUTED \texttt{GOTO} STATEMENT OUTPUT FIRST THE OBJECT WHICH BRANCHES TO SOME PARTICULAR STATEMENT. THE OPERATORS SHOULD BE ABLE TO ACCEPT THESE REQUESTS AS INPUT PARAMETERS AND PROVIDE FACILITIES FOR AS MANY DIFFERENT REQUESTS AS POSSIBLE.

THE OPERATORS IN THE CURRENT ARF ARE NOT CONSTRUCTED AS GENERATORS. BECAUSE OF THIS AND BECAUSE AN OPERATOR MAY BE COMPOSED OF ARBITRARILY MANY REF STATEMENTS, THERE IS THE DANGER THAT APPLICATION OF AN OPERATOR MAY PRODUCE A LARGE NUMBER OF OUTPUT OBJECTS. THIS IS A WEAKNESS IN ARF WHICH RESTRICTS THE RANGE OF PROBLEMS WHICH IT CAN SOLVE. IN CHAPTER VII WE PRESENT REF PROCEDURES (I.E., THE STATEMENT OF THE TOWER OF HANOI PROBLEM AND THE FIRST STATEMENT OF THE PATENTS PROBLEM) WHICH ARF COULD NOT SOLVE BECAUSE OF THIS \texttt{CASE EXPLOSION} PROBLEM.

2. OTHER OPERATOR FORMULATIONS

ONE OF THE REASONS FOR ARF'S TROUBLE WITH CASE EXPLOSIONS IS THAT AN OPERATOR CAN INCLUDE ARBITRARILY MANY REF STATEMENTS. THIS MEANS THAT IF INTERPRETATION OF THE FIRST STATEMENT ON A PATH OF THE DIRECTED GRAPH PRODUCES MORE THAN ONE OUTPUT OBJECT, THEN EACH OF THOSE OBJECTS IS INTERPRETED THROUGH THE REMAINDER OF THE PATH BEFORE THE SEARCH EXECUTIVE HAS AN OPPORTUNITY TO EVALUATE THE NEW OBJECTS AND CONTROL THEIR INTERPRETATION. THIS DIFFICULTY COULD BE
ELIMINATED BY TREATING EACH REF STATEMENT AS AN OPERATOR IN THE
SEARCH SPACE. A SET OF OPERATORS DEFINED IN THIS WAY ARE DESCRIBED
IN FIGURE V.5. THESE OPERATORS ARE SIMILAR TO THE ONES USED BY
ARF; THEY DIFFER ONLY IN THAT THEY RETURN CONTROL TO THE SEARCH
EXECUTIVE AFTER INTERPRETATION OF EACH REF STATEMENT.
OPERATORS FORMED BY SINGLE R Ef STATEMENTS

OPERATOR FOR \texttt{BEGIN}. 
SET AS THE NEXT APPLICABLE OPERATOR IN THE INPUT CONTEXT THE OPERATOR FOR THE STATEMENT AT LINE 2 OF THE PROCEDURE.

OPERATOR FOR \texttt{SET} STATEMENT AT LINE K OF A PROCEDURE. 
CALL THE \texttt{SET} STATEMENT INTERPRETER WITH THE STATEMENT AND THE INPUT OBJECT. SET AS THE NEXT APPLICABLE OPERATOR IN EACH OF THE RESULTING OBJECTS THE OPERATOR FOR THE STATEMENT AT LINE K+1 OF THE PROCEDURE.

OPERATOR FOR STATEMENT \texttt{SET} \texttt{VECTOR X TC Y1,Y2,\ldots,YN} AT LINE K OF A PROCEDURE. 
CALL THE \texttt{SET} STATEMENT INTERPRETER WITH THE STATEMENT \texttt{SET X[I]} TO \texttt{Y1} AND THE INPUT CONTEXT. FOR I=2,3,\ldots,N CALL THE \texttt{SET} STATEMENT INTERPRETER WITH THE STATEMENT \texttt{SET X[I]} TO \texttt{YI} AND EACH OF THE OBJECTS PRODUCED DURING THE INTERPRETATION OF THE \texttt{X[I-1]} \texttt{SET} STATEMENT. SET AS THE NEXT APPLICABLE OPERATOR IN EACH OF THE RESULTING OBJECTS THE OPERATOR FOR THE STATEMENT AT LINE K+1 OF THE PROCEDURE.

OPERATOR FOR \texttt{CONDITION} STATEMENT AT LINE K OF A PROCEDURE. 
ADD THE BOOLEAN EXPRESSION OF THE STATEMENT AS A CONSTRAINT TO THE INPUT OBJECT. SET AS THE NEXT APPLICABLE OPERATOR IN THE RESULTING OBJECT THE OPERATOR FOR THE STATEMENT AT LINE K+1 OF THE PROCEDURE.

OPERATOR FOR STATEMENT \texttt{GO TO X}. 
SET THE OPERATOR FOR STATEMENT X AS THE NEXT APPLICABLE OPERATOR IN THE INPUT OBJECT.

OPERATOR FOR STATEMENT \texttt{IF X THEN Y} AT LINE K OF A PROCEDURE. 

OPERATOR FOR STATEMENT \texttt{GO TO X (Y1,Y2,\ldots,YN)} \texttt{V}. FOR I=1,2,\ldots,N-1 SET \texttt{VX=IV} AS A CONSTRAINT IN A COPY OF THE INPUT OBJECT AND SET THE OPERATOR FOR STATEMENT YI AS THE

FIGURE V.5. OPERATORS FORMED BY SINGLE R Ef STATEMENTS
NEXT APPLICABLE OBJECT. ADD \( \forall x = v \) AS A CONSTRAINT TO THE INPUT OBJECT AND SET THE OPERATOR FOR STATEMENT \( \text{YN} \) AS THE NEXT APPLICABLE OPERATOR IN THE RESULTING OBJECT.

OPERATOR FOR \( \text{VENDV} \).

CONDUCT THE BACKTRACKING SEARCH TO FIND A SET OF VARIABLE VALUES WHICH SATISFY THE CONSTRAINTS IN THE INPUT OBJECT. IF THE SEARCH IS SUCCESSFUL, THEN OUTPUT AN OBJECT IN WHICH THE VALUES ARE ASSIGNED TO THE VARIABLES. IF THE SEARCH FAILS, PRODUCE NO OUTPUT OBJECT.

FIGURE V.5 (CONTINUED).
The methods used by ARPS search executive are not strong enough to make effective use of the extra control that single statement operators would provide. If the search methods were strengthened, then their effectiveness would depend on having control after the interpretation of each REF statement.

Whenever ARF adds a constraint to a context during the application of an operator (i.e., during interpretation) all the constraint manipulation methods are applied to the context and the new constraint. As we observed in Chapter IV this is an inefficient way of solving a context's constraint satisfaction problem because the order in which constraints are processed is dependent on the order in which they are created. The motivation for designing ARF so that new constraints are processed immediately is that contexts containing inconsistencies are eliminated from the search tree by this processing. ARPS search executive uses this elimination capability to prune the search tree by extending those nodes of the tree which have the highest probability of being eliminated.

The question arises as to whether the LCSS of efficiency in solving the constraint satisfaction subproblems is made up for by the elimination of contexts from the search tree. Clearly this depends on the problem being solved and therefore should be under
THE CONTROL OF ARFS SEARCH EXECUTIVE. THIS COULD BE DONE BY MAKING THE CONSTRAINT MANIPULATION METHODS OPERATORS IN THE SEARCH SPACE. FIGURE V.6 DESCRIBES A SET OF OPERATORS WHICH COULD BE DEFINED FROM THE CONSTRAINT MANIPULATION METHODS.

THE FIRST THREE OF THESE OPERATORS CORRESPOND TO THE THREE PHASES OF CONSTRAINT MANIPULATION DESCRIBED IN CHAPTER IV. THEY INPUT A CONSTRAINT AND A CONTEXT AND CARRY OUT ONE PHASE OF THE PROCESSING ON THE INPUT CONSTRAINT. WE ARE ASSUMING THAT EACH OF A CONTEXT'S CONSTRAINTS IS MARKED AS AVAILABLE FOR INPUT TO ONE OF THE PHASE OPERATORS OR IS MARKED PROCESSED. WHEN A NEW CONSTRAINT IS ADDED TO A CONTEXT IT IS MARKED AS AVAILABLE FOR INPUT TO THE PHASE 1 OPERATOR. GIVEN THESE OPERATORS THE SEARCH EXECUTIVE CAN CONTROL WHEN OR IF EACH PHASE OF PROCESSING IS APPLIED TO EACH CONSTRAINT IN EACH CONTEXT.
OPERATORS FORMED BY THE CONSTRAINT SATISFACTION METHODS

PHASE 1 OPERATOR
INPUT A CONTEXT AND A CONSTRAINT. DO THE PHASE 1 REDUCTION OF THE INPUT CONSTRAINT AND TEST FOR REDUCTION TO \( \text{\texttt{TRUE}} \) OR \( \text{\texttt{FALSE}} \). IF THE CONSTRAINT DOES NOT REDUCE TO \( \text{\texttt{TRUE}} \) OR \( \text{\texttt{FALSE}} \), THEN MARK IT AS AVAILABLE FOR INPUT TO THE PHASE 2 OPERATOR. IF THE CONSTRAINT DOES NOT REDUCE TO \( \text{\texttt{FALSE}} \), THEN OUTPUT THE RESULTING CONTEXT; OTHERWISE PRODUCE NO OUTPUT.

PHASE 2 OPERATOR
INPUT A CONTEXT AND A CONSTRAINT. DO THE PHASE 2 ACTIONS IN AN ATTEMPT TO MAKE DEDUCTIONS FROM THE INPUT CONSTRAINT. IF NO INCONSISTENCY IS FOUND AND THE CONSTRAINT IS NOT ELIMINATED, THEN MARK THE CONSTRAINT AS AVAILABLE FOR INPUT TO THE PHASE 3 OPERATOR. IF NO INCONSISTENCY IS FOUND, THEN OUTPUT THE RESULTING CONTEXT; OTHERWISE PRODUCE NO OUTPUT.

PHASE 3 OPERATOR
INPUT A CONTEXT AND A CONSTRAINT. DO THE PHASE 3 ACTIONS IN AN ATTEMPT TO MAKE DEDUCTIONS CONSIDERING THE INPUT CONSTRAINT IN CONJUNCTION WITH THE CONTEXT'S PROCESSED CONSTRAINTS. IF NO INCONSISTENCY IS FOUND AND THE CONSTRAINT IS NOT ELIMINATED, THEN MARK THE CONSTRAINT PROCESSED. IF NO INCONSISTENCY IS FOUND, THEN OUTPUT THE RESULTING CONTEXT; OTHERWISE PRODUCE NO OUTPUT.

ASSIGN OPERATOR
INPUT A CONTEXT, A VARIABLE NAME, AND AN ELEMENT FROM THE RANGE OF THE INPUT VARIABLE. LET \( s(i) \) DENOTE THE INPUT VARIABLE AND \( k \) DENOTE THE INPUT RANGE ELEMENT. CALL THE PHASE 2 OPERATOR WITH THE CONSTRAINT \( \forall s(i) = k \) AND A COPY OF THE INPUT CONTEXT AS INPUTS. CALL THE PHASE 2 OPERATOR WITH THE CONSTRAINT \( \forall s(i) = k \) AND THE INPUT CONTEXT AS INPUTS. OUTPUT THE RESULTING CONTEXTS.

FIGURE V.2. OPERATORS FROM THE CONSTRAINT SATISFACTION METHODS
THE FOURTH OPERATOR SHOWN IN FIGURE V.6, THE ASSIGN OPERATOR, PERFORMS THE VALUE ASSIGNMENTS THAT OCCUR DURING THE BACKTRACKING SEARCH. THAT IS, GIVEN A CONTEXT, A VARIABLE, AND AN ELEMENT OF THE VARIABLE'S RANGE, IT DELETES THE RANGE ELEMENT FROM THE RANGE OF THE VARIABLE IN THE CONTEXT AND ASSIGNS THE RANGE ELEMENT AS THE VALUE OF THE VARIABLE IN A COPY OF THE CONTEXT. THIS OPERATOR MAY BE APPLIED TO ANY CONTEXT WHICH HAS AN UNVALUED VARIABLE. SINCE WITH THIS OPERATOR THE SEARCH EXECUTIVE CAN ASSIGN VALUES TO VARIABLES IN ANY CONTEXT AT ANY TIME DURING THE INTERPRETATION, THERE WOULD NO LONGER BE ANY NEED FOR THE VENDV STATEMENT OPERATOR TO CONDUCT A SEPARATE BACKTRACKING SEARCH.

GIVEN THESE OPERATORS DERIVED FROM THE CONSTRAINT SATISFACTION METHODS, WE MAY FORMULATE THE SEARCH PROBLEM SO THAT ALL THE PROGRAM'S PROBLEM SOLVING ACTIVITIES ARE UNDER THE CONTROL OF A SINGLE SEARCH EXECUTIVE ROUTINE. THE OPERATORS IN THIS FORMULATION ARE THOSE OF FIGURES V.5 AND V.6, WITH THE EXCEPTION THAT THE OPERATOR FOR THE VENDV STATEMENT IN FIGURE V.5 IS ALTERED SO THAT IT DOES NOTHING MORE THAN OUTPUT ITS INPUT OBJECTS AFTER MARKING EACH OF THEM AS HAVING BEEN INTERPRETED TO THE VENDV STATEMENT. THE GOAL OF THE SEARCH EXECUTIVE IS TO PRODUCE AN OBJECT WHICH HAS BEEN INTERPRETED TO THE VENDV STATEMENT AND WHICH HAS AN INTEGER VALUE FOR EACH OF ITS VARIABLES. NOTE THAT WITH
THESE ADDITIONAL OPERATORS IT IS NO LONGER THE CASE THAT ONLY ONE OPERATOR IS APPLICABLE TO AN OBJECT AT A GIVEN TIME. THE EXECUTIVE MAY BE ABLE TO APPLY A REP STATEMENT OPERATOR TO AN OBJECT, A CONSTRAINT MANIPULATION OPERATOR TO ANY ONE OF AN OBJECT'S UNPROCESSED CONSTRAINTS, OR THE ASSIGN OPERATOR TO ANY RANGE ELEMENT OF ANY OF AN OBJECT'S UNVALUED VARIABLES.

THIS FORMULATION OF THE SEARCH PROBLEM PROVIDES THE SEARCH EXECUTIVE WITH A MAXIMUM OF CONTROL OVER THE ACTIVITIES OF THE PROGRAM. IT CAN FREELY INTERMIX INCREMENTAL STEPS OF INTERPRETATION, CONSTRAINT PROCESSING, AND VALUE ASSIGNMENT TO VARIABLES. THE FLEXIBILITY AND CENTRALIZED CONTROL OF THIS FORMULATION WOULD GREATLY FACILITATE THE DESIGN OF PROBLEM SOLVING METHODS FOR MONITORING, EVALUATING, AND GUIDING THE SEARCH.

B. METHODS FOR DIRECTING THE SEARCH

THE PROBLEM OF SEARCH IN A SPACE OF OPERATORS AND OBJECTS IS ONE WHICH OCCURS IN MOST ARTIFICIAL INTELLIGENCE PROGRAMS AND MUCH IS KNOWN ABOUT GUIDING AND LIMITING THE SEARCH. NEWELL AND ERNST (1965) HAVE PRESENTED A GENERAL FORMULATION BOTH OF THE HEURISTIC SEARCH PROBLEM AND OF THE DEVICES USED IN PROBLEM SOLVERS FOR DOING THE SEARCH. THESE DEVICES INCLUDE OBJECT EVALUATION FUNCTIONS, MEANS-ENDS ANALYSIS, OPERATOR ORDERING, AVOIDANCE OF
GENERATING AN OBJECT ALREADY OBTAINED, PLANNING, ETC. IN THIS SECTION WE WILL INVESTIGATE WHICH OF THESE DEVICES CAN BE EFFICIENTLY IMPLEMENTED FOR OUR PARTICULAR SEARCH PROBLEM. WE BEGIN WITH A DESCRIPTION OF THE DEVICES WHICH EXIST IN THE CURRENT ARF PROGRAM AND THEN EXPLORE THE DESIGN OF POSSIBILITIES FOR IMPLEMENTING ADDITIONAL DEVICES TO IMPROVE ITS SEARCHING CAPABILITIES.

1. ARFs SEARCH STRATEGY

THE EXECUTIVE ROUTINE IN ARF WHICH GUIDES THE SEARCH IN THE SUBPROBLEM SPACE IS FLOW CHARTED IN FIGURE V.7. AN OBJECT EVALUATION ROUTINE IS USED TO SELECT AN OBJECT FOR EACH STEP IN THE SEARCH, AND THE INTERPRETER IS USED TO APPLY AN OPERATOR TO THE SELECTED OBJECT.
Figure V.7. CIF's top executive
THE OBJECT EVALUATION ROUTINE USES AN ALGORITHM WHICH ATTEMPTS TO SELECT THE EASIEST SUBPROBLEM AT EACH STEP AND TO GUARD AGAINST BEING TRAPPED IN LOOPS. THE EASE OF A SUBPROBLEM IS MEASURED IN TERMS OF ITS SOLUTION SPACE SIZE. THE SOLUTION SPACE SIZE OF A SUBPROBLEM HAVING N VARIABLES, WHERE EACH VARIABLE S(I) HAS R(I) ELEMENTS IN ITS RANGE, IS CONSIDERED TO BE THE PRODUCT R(1)*R(2)*...*R(N). THE SUBPROBLEMS WITH THE SMALLEST SOLUTION SPACES ARE CONSIDERED TO BE THE EASIEST BECAUSE THEY USUALLY PRODUCE THE LEAST NUMBER OF NEW SUBPROBLEMS AND HAVE THE GREATEST PROBABILITY OF BEING ELIMINATED BY A CONTRADICTION DURING FURTHER INTERPRETATION. HENCE, BY DEALING WITH THE EASIEST SUBPROBLEMS FIRST A&P REDUCES THE NUMBER OF SUBPROBLEMS WHICH IT MUST REMEMBER AT ANY GIVEN TIME. THIS, IN TURN, REDUCES BOTH THE MEMORY REQUIREMENTS OF THE SYSTEM AND THE TIME REQUIRED TO DECIDE WHICH SUBPROBLEM TO CONSIDER NEXT AFTER EACH INTERPRETATION STEP.

TO SEE THE EFFECT OF THIS SELECTION ALGORITHM CONSIDER THE PROBLEM SHOWN IN FIGURES V.8A AND V.8B. IT IS A FORMALIZATION USED BY MANFRED KOCHEN (1960) OF A TYPICAL TYPE OF CONCEPT FORMATION AND HYPOTHESIS FORMATION PROBLEM FOUND IN PSYCHOLOGICAL EXPERIMENTS. THE E(I) REPRESENT EXEMPLARS HAVING THREE BINARY ATTRIBUTES. THE SET H REPRESENTS A DISJUNCTIVE HYPOTHESIS WHICH WILL APPROPRIATELY CLASSIFY EACH EXEMPLAR. THE REP STATEMENT OF
THIS PROBLEM BEGINS BY DEFINING A VECTOR \( \mathbf{H} \) AND THE EIGHT EXEMPLARS. A SELECTION IS MADE TO DETERMINE HOW MANY \( H(i) \) SETS WILL BE IN THE SOLUTION, AND EACH \( H(i) \) IS DEFINED BY THE SELECTION OF A TRIPLE. VERIFICATION OF THE SELECTED SOLUTION IS CARRIED OUT IN THE L2 LOOP BY CONSIDERING EACH EXEMPLAR. THE L3 LOOP TESTS EACH \( H(j) \) TO DETERMINE IF IT CONTAINS THE CURRENT EXEMPLAR. IF AN EXEMPLAR IS AN ELEMENT OF ONE OF THE SETS \( H(j) \), THEN ITS FOURTH MEMBER MUST BE 1; OTHERWISE ITS FOURTH ELEMENT MUST BE 0.
PROBLEM:

THE FOUR-TUPLES E1, ..., E8 ARE DEFINED AS FOLLOWS:

\[
\begin{align*}
E1: & \quad 0, 1, 0, Y \\
E2: & \quad 0, 1, 1, N \\
E3: & \quad 1, 1, 0, Y \\
E4: & \quad 1, 1, 1, Y \\
E5: & \quad 0, 0, 0, W \\
E6: & \quad 0, 0, 1, W \\
E7: & \quad 1, 0, 0, Y \\
E8: & \quad 1, 0, 1, Y
\end{align*}
\]

FIND SETS H(1), H(2), ..., H(M) (1 \leq M \leq 5) SUCH THAT:

1) EACH E(I) IS AN ELEMENT OF THE SET H(J) IF AND ONLY IF ITS FOURTH MEMBER IS Y.

2) EACH H(J) IS A SUBSET OF THE SET WHOSE ELEMENTS ARE THE TRIPLES (0, 0, 0), (0, 0, 1), (0, 1, 0), (0, 1, 1), ..., (1, 1, 1). ANY H(J) MAY BE DEFINED BY A TRIPLE (Z1, Z2, Z3), WHERE Z(K) IS 0, 1, OR 2, SUCH THAT FOR EACH Z(K) EQUAL TO 0 OR 1 A TRIPLE (X1, X2, X3) IS AN ELEMENT OF H(J) IF AND ONLY IF X(K) = Z(K).

3) EACH E(I) IS AN ELEMENT OF H(J) IF AND ONLY IF THE TRIPLE FORMED BY ITS FIRST THREE MEMBERS IS AN ELEMENT OF H(J).

SOLUTION:

H = (2, 1, 0) \quad (1, 2, 2).

FIGURE V.8A. ENGLISH STATEMENT OF A HYPOTHESIS FORMATION PROBLEM
BEGIN;

SET_VECTOR H TO H1, H2, H3, H4, H5;
SET_VECTOR E TO E1, E2, E3, E4, E5, E6, E7, E8;
SET_VECTOR E1 TO 0, 1, 0, Y;
SET_VECTOR E2 TO 0, 1, 1, N;
SET_VECTOR E3 TO 1, 1, 0, Y;
SET_VECTOR E4 TO 1, 1, 1, Y;
SET_VECTOR E5 TO 0, 0, 0, N;
SET_VECTOR E6 TO 0, 0, 1, N;
SET_VECTOR E7 TO 1, 0, 0, Y;
SET_VECTOR E8 TO 1, 0, 1, Y;
SET <M> TO SELECT(1, 5);
FOR I = <M> DO TO L1;
L1: SET_VECTOR H[<I>] TO SELECT(0, 2), SELECT(0, 2), SELECT(0, 2);
FOR J = <M> DO TO L3;
FOR K = 3 DO TO L4;
IF ~(H[<J>][<K>] = 2) ^ ~(H[<J>][<K>] = E[<I>][<K>]) THEN L3;
L4:;
CONDITION E[<I>][4] = Y;
GOTO L2;
L3:;
CONDITION E[<I>][4] = N;
L2:;
END;

FIGURE V. 8B. REP STATEMENT OF A HYPOTHESIS FORMATION PROBLEM
Figure V.9. First portion of IP's search tree for the hypothesis formation problem.
The selection algorithm of the object evaluation routine leaves open the possibility that interpretation will become trapped in a loop where the solution space size of the selected subproblem does not increase and the same subproblem is continually reselected. This problem occurs most frequently during the interpretation of heuristic search problems such as the monkey and waterjug problems discussed in Chapter III. For these problems one would like AIF to find the solution having the least number of selections, but since all the subproblems in the waterjug problem have solution spaces of size one and the monkey problem contains loops in which a subproblem's solution space does not grow, selection criteria other than solution space size are needed to guard against looping.

We have responded to this problem by including in AIF's selection routine the provision that a subproblem having k variables can be selected only if there are no other subproblems having less than k-1 variables. This restriction prevents consideration of a subproblem which is two selections beyond the other subproblems and thereby insures some consideration of all subproblems.

During interpretation where new variables are continually being added, this selection routine produces a basically breadth...
FIRST SEARCH BEHAVIOR. THE SOLUTION SPACE SIZE CRITERION PROVIDES
SELECTIVITY AT EACH STEP, BUT THE RESTRICTION ON THE NUMBER OF
VARIABLES INSURES THAT NO TERMINAL NCDE OF THE TREE IS MORE THAN
TWO LEVELS BEYOND ANY OTHER TERMINAL NCDE. FIGURE V. 10 SHOWS THE
TREE PRODUCED BY THIS BREADTH FIRST SEARCH BEHAVIOR FOR THE MONKEY
PROBLEM.

DURING INTERPRETATION WHERE NO NEW VARIABLES ARE BEING ADDED,
ARFS SELECTION ROUTINE PRODUCES DEPTH FIRST SEARCHING. THAT IS,
ONCE A SUBPROBLEM IS SELECTED FOR INTERPRETATION, THE SELECTION
ROUTINE WILL CONTINUE TO SELECT IT OR SUBPROBLEMS PRODUCED FROM IT
AS LONG AS IT AND ANY OF ITS SUBPROBLEMS EXIST. THERE ARE TWO
REASONS FOR THIS BEHAVIOR. FIRST, SINCE NO NEW VARIABLES ARE BEING
CREATED THE RESTRICTION ON THE NUMBER OF VARIABLES IN A CONTEXT
SELECTED FOR INTERPRETATION DOES NOT AFFECT THE SEARCH. SECOND,
WHEN A SUBPROBLEM IS DIVIDED INTO NEW SUBPROBLEMS DURING
INTERPRETATION THE SOLUTION SPACES OF THE NEW SUBPROBLEMS CANNOT
BE LARGER THAN THE SOLUTION SPACE OF THE PARENT SUBPROBLEM. THIS
DEPTH-FIRST SEARCH BEHAVIOR OCCURS FOR THE HYPOTHESIS FORMATION
PROBLEM AND IS ILLUSTRATED IN FIGURE V. 9.
Figure V.10. Search tree produced by ARF for the monkey problem
WE SEE THAT ARFVS SEARCH BEHAVIOR DEVIATES ONLY SLIGHTLY FROM THAT WHICH WOULD BE PRODUCED BY A FIXED ORDER EXHAUSTIVE SEARCH ALGORITHM. IN THE REMAINDER OF THIS CHAPTER WE WILL EXPLORE VARIOUS POSSIBILITIES FOR IMPROVING ARFVS SEARCH CAPABILITY. WE WILL CONSIDER DEVICES FOR DETECTING WHEN TWO SUBPROBLEMS ARE EQUIVALENT, FOR CONDUCTING A BI-DIRECTIONAL SEARCH, FOR INCREASING THE EFFECTIVENESS OF THE CONTEXT SELECTION ROUTINE, AND FOR DOING MEANS-ENDS ANALYSIS.

2. RECOGNITION OF LOOPS

ONE IMPORTANT WAY OF IMPROVING A PROBLEM SOLVERVS ABILITY TO SEARCH THROUGH A SPACE IS TO GIVE IT THE CAPABILITY OF ANSWERING THE QUESTION "HAVE I CONSIDERED THIS POINT IN THE SPACE BEFORE?" BECAUSE ARF LACKS THIS CAPABILITY, FINDING A SOLUTION TO THE WATERJUG PROBLEM IS INFEASIBLE AND FINDING A SOLUTION TO THE MONKEY PROBLEM IS LONG AND TEDIOUS. FOR ARF TO HAVE THIS CAPABILITY IT MUST DO MORE THAN JUST KEEP A RECORD OF WHERE IT HAS BEEN, BECAUSE TWO CONTEXTS NEED NOT BE IDENTICAL TO REPRESENT EQUIVALENT STATES (IN FACT, IDENTICAL CONTEXTS ARE RARELY PRODUCED). FOR EXAMPLE, CONSIDER THE TWO SUBPROBLEMS OF THE MONKEY PROBLEM SHOWN IN FIGURE V.11. THESE NONIDENTICAL SUBPROBLEMS REPRESENT EQUIVALENT STATES IN THE SENSE THAT THE BOX IS IN THE
SAME POSITION AND THE POSSIBLE POSITIONS OF THE MONKEY ARE THE SAME IN BOTH SUBPROBLEMS. ONE WOULD LIKE ARP TO RECOGNIZE THIS EQUIVALENCE AND DELETE CONTEXT 15204.
CONTEXT 14831

DATA STRUCTURE
BOX
  VECTOR: X2, ON.FLOOR
MONKEY
  VECTOR: X[S(1)], ON.FLOOR
Y
  VECTOR: ON.FLOOR, ON.BOX
X
  VECTOR: X1, X2, UNDER.BANANAS

VARIABLES
S(1)
  RANGE: 1, 3

CONSTRAINTS
NONE

CONTEXT 15204

DATA STRUCTURE
BOX
  VECTOR: X2, ON.FLOOR
MONKEY
  VECTOR: X[S(2)], ON.FLOOR
Y
  VECTOR: ON.FLOOR, ON.BOX
X
  VECTOR: X1, X2, UNDER.BANANAS

VARIABLES
S(2)
  RANGE: 1, 3
S(1)
  RANGE: 1, 3

CONSTRAINTS
NONE

FIGURE V.11. TWO NONIDENTICAL EQUIVALENT CONTEXTS
IF WE ASSUME A VERSION OF ARF WHICH STORES A COPY OF EACH CONTEXT PRODUCED DURING INTERPRETATION, THEN WHAT ARE THE NECESSARY AND SUFFICIENT CONDITIONS FOR A NEW CONTEXT PRODUCED BY THE INTERPRETER TO BE ELIMINATED AS REDUNDANT. TO STATE THESE CONDITIONS WE NEED THE FOLLOWING DEFINITION. AN ADMISSIBLE VALUE ASSIGNMENT FOR A CONTEXT IS AN ASSIGNMENT OF A VALUE TO EACH VARIABLE OF THE CONTEXT SUCH THAT EACH VARIABLE'S VALUE IS AN ELEMENT OF ITS RANGE AND THE ASSIGNMENT SATISFIES THE CONTEXT'S CONSTRAINTS. GIVEN THIS DEFINITION A NEW CONTEXT X PRODUCED BY THE INTERPRETER IS REDUNDANT WITH RESPECT TO THE CONTEXTS PREVIOUSLY PRODUCED BY THE INTERPRETER IF AND ONLY IF FOR EVERY ADMISSIBLE VALUE ASSIGNMENT FOR CONTEXT X THERE EXISTS A PREVIOUSLY PRODUCED CONTEXT Y AND AN ADMISSIBLE VALUE ASSIGNMENT FOR CONTEXT Y SUCH THAT THE ASSIGNMENT MAKE THE DATA STRUCTURES OF CONTEXTS X AND Y IDENTICAL (NOTE: A CONTEXT'S NEXT APPLICABLE OPERATOR IS CONSIDERED PART OF ITS DATA STRUCTURE).

THE USE OF A GENERAL TEST FOR DETERMINING WHEN A NEW CONTEXT IS REDUNDANT WOULD IN MOST CASES REQUIRE MORE PROCESSOR TIME TO APPLY THAN IT WOULD YIELD IN SEARCH REDUCTION. AN ALTERNATIVE COURSE OF ACTION MIGHT BE TO USE A TEST WHICH LOOKS ONLY FOR SOME SET OF SUFFICIENT CONDITIONS FOR ELIMINATING A CONTEXT. HOPEFULLY SUCH A TEST WOULD DETECT OBVIOUS LOOPING SITUATIONS AND WOULD BE
INEXPENSIVE TO APPLY.

CONSIDER THE FOLLOWING CANDIDATE FOR SUCH A TEST. WHEN A NEW CONTEXT IS PRODUCED IT IS SUCCESSIVELY COMPARED WITH EACH PREVIOUSLY PRODUCED CONTEXT. FOR EACH PAIRWISE COMPARISON, SUCCESS OF THE TEST INDICATES THAT THE NEW CONTEXT IS REDUNDANT AND CAN BE ELIMINATED; FAILURE OF THE TEST INDICATES THAT REDUNDANCE CANNOT BE DETERMINED. IF EITHER OF THE CONTEXTS HAS CONSTRAINTS OR NONINTEGER VARIABLE VALUES, THEN THE TEST FAILS. THE TEST SUCCEEDS ONLY IF A MAPPING $f$ CAN BE FOUND WHICH SATISFIES THE FOLLOWING CONDITIONS:

1. $f$ MAPS THE VARIABLES OF THE OLD CONTEXT INTO THE VARIABLES OF THE NEW CONTEXT ($f$ NEED NOT BE ONE-TO-ONE).

2. IF $s(i)$ IS A VARIABLE IN THE OLD CONTEXT AND $j$ IS IN THE RANGE OF $f(s(i))$, THEN $j$ IS ALSO IN THE RANGE OF $s(i)$.

3. IF EVERY OCCURRENCE OF EACH VARIABLE $s(i)$ IN THE DATA STRUCTURE OF THE OLD CONTEXT IS REPLACED BY $f(s(i))$, THEN EVERY VALUE EXPRESSION IN THE DATA STRUCTURE OF THE OLD CONTEXT IS IDENTICAL TO THE CORRESPONDING VALUE EXPRESSION IN THE DATA STRUCTURE OF THE NEW CONTEXT.

THE EXISTENCE OF SUCH AN $f$ ASSURES THAT FOR ALL ADMISSIBLE VALUE
ASSIGNMENTS FOR THE NEW CONTEXT THERE EXISTS AN ADMISSIBLE VALUE ASSIGNMENT FOR THE OLD CONTEXT SUCH THAT THE ASSIGNMENTS WILL MAKE THE DATA STRUCTURES OF THE TWO CONTEXTS IDENTICAL.

THIS TEST WOULD BE SUFFICIENT TO RECOGNIZE THE LOOFING IN THE MONKEY PROBLEM AND IN THE WATERJUG PROBLEM. FIGURE V.12 SHOWS THE SEARCH TREE PRODUCED BY ARF WHILE SOLVING THE MONKEY PROBLEM AND INDICATES THE PORTION OF THAT SEARCH WHICH COULD HAVE BEEN AVOIDED BY THE USE OF THE LOOPING TEST. THE TEST IS WEAK IN THAT IT EXCLUDES FROM CONSIDERATION CONTEXTS HAVING CONSTRAINTS, BUT FOR PROBLEMS WHICH REQUIRE THE PRODUCTION OF MANY SUBPROBLEMS IT IS OFTEN THE CASE THAT THE SUBPROBLEMS HAVE NO CONSTRAINTS. THE CONSTRAINTS IN EACH SUBPROBLEM ARE SIMPLE ENOUGH TO BE ELIMINATED BY THE CONSTRAINT MANIPULATION METHODS.
Figure V.12. Indication of the portion of ARF's search for the solution to the monkey problem which could have been avoided by the use of the proposed loop recognition algorithm
3. USE OF AN INVERSE INTERPRETER

Another of ARP's major weaknesses is its lack of ability to do any means-ends analysis during its search. If ARP could determine which of its contexts was **closest** to the **END** statement, then it could choose that context for continued interpretation. Similarly, if ARP knew what had to be true in a context for the interpreter to be able to move it to the **END** statement, then it might set up a subgoal to interpret the context through a particular path in the procedure which contained statements that would make the desired change. We will describe in the following paragraphs an inverse interpreter for REF procedures which can provide the basis for building these desired features into ARP. This interpreter has been programmed but is not a functioning part of the current ARP.

An inverse interpreter moves contexts through a REF procedure as does the standard interpreter, but it does so in the opposite direction. For example, inverse interpretation of a **SET** statement means starting with a context in which the assignment indicated by the **SET** statement is assumed to have been made and modifying the context so that it is in a state equivalent to the state it was in before the assignment was made. As the inverse interpreter moves from **END** towards **BEGIN**, it uses contexts,
VARIABLES, AND CONSTRAINTS AND CREATES SUBPROBLEMS AS DOES THE
STANDARD INTERPRETER. WE WILL NOW GIVE A BRIEF DESCRIPTION OF HOW
INVERSE INTERPRETATION IS DEFINED FOR EACH REF STATEMENT.

INVERSE INTERPRETATION OF AN \texttt{IFV} OR COMPUTED \texttt{GOTOV}
STATEMENT INVOLVES ADDING THE CONSTRAINT TO THE CONTEXT WHICH MUST
HAVE BEEN TRUE FOR THE STANDARD INTERPRETER TO HAVE TAKEN THE PATH
ON WHICH THE INVERSE INTERPRETER IS APPROACHING THE \texttt{IFV} OR
COMPUTED \texttt{GOTOV} STATEMENT. FOR EXAMPLE, IF IN THE MONKEY PROBLEM
THE INVERSE INTERPRETER MOVES A CONTEXT FROM \texttt{GET EANANAS} THROUGH
THE COMPUTED \texttt{GOTOV} STATEMENT OF LINE 13, IT WILL ADD TO THE
CONTEXT THE CONSTRAINT \texttt{<M>=1v}.

WHEN THE INVERSE INTERPRETER COMPLETES INTERPRETATION OF A
LABELED STATEMENT WHICH CAN FOLLOW MORE THAN ONE STATEMENT DURING
FORWARD INTERPRETATION (I.E. A \texttt{JCN ECINT}), IT SAVES A COPY OF THE
CONTEXT AND CONSIDERS EACH PATH WHICH THE FORWARD INTERPRETER
COULD HAVE TAKEN TO REACH THE LABELED STATEMENT AS A SEPARATE
CASE. THIS CASE ANALYSIS CREATES A SUBPROBLEM SPACE TO BE SEARCHED
AS IT DOES DURING FORWARD INTERPRETATION. NOTE THAT THE INVERSE
INTERPRETER NEED NOT ADD A BRANCHING CONDITION TO EACH NEW
SUBPROBLEM AS DOES THE STANDARD INTERPRETER.

INVERSE INTERPRETATION OF A \texttt{SETV} STATEMENT IS SIMILAR TO
FORWARD INTERPRETATION. WHEN THE INVERSE INTERPRETER ENCOUNTERS A
\texttt{vsetv} statement it can deduce that all references in the context to the data slot defined by the statement\texttt{vs} slot expression can be replaced by the statement\texttt{vs} right side expression. For example, in the monkey problem consider the context referred to above which was moved from \texttt{vget.bananas} through the computed \texttt{vgoto\_v} statement of line 13. When that context is interpreted through the \texttt{vsetv} statement of line 12, each occurrence of \texttt{<m>} in a constraint, data slot value, or variable value will be replaced by the variable defined by the \texttt{vselectv} function call in the right side of the \texttt{vsetv} statement. In particular, the constraint \texttt{<m>=1\_v} added at the computed \texttt{vgoto\_v} statement will become \texttt{vs(i)=1\_v}. The inverse interpreter assigns identifying integers to variables in descending order beginning with the arbitrarily chosen value of 99 so that the variables \texttt{s(i)}, \texttt{s(i+1)}, ..., \texttt{s(99)} in a context denote the last 100\_i selections to be made in forming the solution to the subproblem represented by the context.

In the process of replacing the references to the data slot indicated by the \texttt{vsetv} statement\texttt{vs} slot expressions, the inverse interpreter may have to carry out the same kind of matching operations required during the forward interpretation of a \texttt{vsetv} statement. The \texttt{vsetv} statement processor used by the forward interpreter (described in Chapter VI) is organized so that it can be used by the inverse interpreter. The \texttt{\_vreplace\_v} subroutine,
WHICH IS A PART OF THAT PROCESSOR, CAN DO THE REPLACEMENTS REQUIRED BY THE INVERSE INTERPRETATION OF A \texttt{SETV} STATEMENT.

\section*{3.1. Conducting a Two Way Search}

Given such an inverse interpreter, consider how it might be used to assist in AFS's search through a subproblem space. The most direct way in which it could be used is to conduct a two way search, using the standard interpreter to grow a search tree from \texttt{BEGIN\texttt{v} TOWARD \texttt{VEND\texttt{v}}} and the inverse interpreter to grow a second search tree from \texttt{VEND\texttt{v} TOWARD \texttt{BEGIN\texttt{v}}}.

For a two way search to be more effective than a one way search the problem solver needs an inexpensive way of determining if any context at a terminal node of one tree can be combined with a context at a terminal node of the other tree so that the combined context has, in effect, been interpreted from \texttt{BEGIN\texttt{v}} to \texttt{VEND\texttt{v}}. That is, the problem solver must be able to determine when the trees can be joined to complete the search.

The following must be true about the context formed by combining a \texttt{\texttt{v}v\texttt{FORWARD\texttt{v}}} context (generated by the standard interpreter) and a \texttt{\texttt{v}v\texttt{BACKWARD\texttt{v}}} context (generated by the inverse interpreter):

1. The combined context has the variables of both the
FORWARD AND THE BACKWARD CONTEXTS, WHERE IT IS
ASSUMED THAT NO VARIABLE NAME APPEARS IN BOTH
CONTEXTS.

2. THE COMBINED CONTEXT HAS THE CONSTRAINTS OF BOTH THE
FORWARD AND BACKWARD CONTEXTS.

3. ANY DATA SLOT WHICH HAS A VALUE IN EITHER THE FORWARD
OR BACKWARD CONTEXT HAS AN IDENTICAL OR EQUIVALENT
VALUE IN THE COMBINED CONTEXT.

IF A COMBINED CONTEXT CAN BE FORMED WHICH HAS THESE
CHARACTERISTICS, THEN IT REPRESENTS A CONTEXT WHICH HAS BEEN
INTERPRETED FROM VEEGINV TO VENDV AND ANY SET OF CONSISTENT VALUES
FOR ITS VARIABLES REPRESENTS A SOLUTION TO THE ORIGINAL PROBLEM.

CONSIDER THE USE OF SUCH A TWO WAY SEARCH IN SOLVING THE
MONKEY PROBLEM. ASSUME A SIMPLE BREADTH-FIRST SEARCH STRATEGY AS
FOLLOWS. EACH TIME AN INTERPRETER IS CALLED WITH A SINGLE CONTEXT
AS INPUT IT CONTINUES INTERPRETING UNTIL IT PRODUCES MORE THAN ONE
CONTEXT AS OUTPUT (SUCH AS AT A BRANCH OR JOIN POINT). WHENEVER A
FORWARD AND A BACKWARD CONTEXT ARE AT THE SAME LOCATION IN THE REF
PROCEDURE, AN ATTEMPT IS MADE TO COMBINE THEM. IF THE COMBINATION
ATTEMPT FAILS OR IF IT SUCCCEEDS AND AN ACCEPTABLE SET OF VARIABLE
VALUES FOR THE COMBINED CONTEXT CANNOT BE FOUND, THEN PROCESSING
CONTINUES AS IF THE COMBINATION HAD NOT BEEN TRIED. AT EACH STEP
IN THE SEARCH THE EXECUTIVE CHOOSES WHICH SEARCH TREE TO EXTEND
AND THEN USES THE APPROPRIATE INTERPRETER TO INTERPRET EACH CONTEXT AT A TERMINAL NODE OF THE TREE. THE EXECUTIVE CHOOSES FOR EXTENSION THE SEARCH TREE WHICH HAS THE FEWEST TERMINAL NODES IN ORDER TO MINIMIZE THE NUMBER OF SUBPROBLEMS GENERATED DURING A SOLUTION PROCESS.

THE SEARCH TREES THIS STRATEGY WOULD PRODUCE FOR THE MONKEY PROBLEM ARE SHOWN IN FIGURE V.13. ASSUMING THAT THE STANDARD INTERPRETER IS CALLED FIRST, IT INTERPRETS TO THE VIIFV STATEMENT AT LINE 7 OF THE REF PROBLEM STATEMENT. THE INVERSE INTERPRETER IS THEN CALLED AND INTERPRETS TO STATEMENT L1. AT THIS POINT THE FORWARD TREE WOULD BE CONSIDERED TO HAVE TWO TERMINAL NODES (REPRESENTING THE TWO SUBPROBLEMS PRODUCED AT THE LINE 7 VIIFV STATEMENT) AND THE BACKWARD TREE TO HAVE THREE TERMINAL NODES (REPRESENTING THE TWO SUBPROBLEMS PRODUCED AT STATEMENT L1). THE EXECUTIVE CHOOSES THE FORWARD TREE FOR EXTENSION. THE STANDARD INTERPRETER INTERPRETS THE SUBPROBLEM WHICH ERANCHED TO CVWALKV AT LINE 7 THROUGH THE VSETV STATEMENT AT CVWALKV AND BACK TO THE VIIFV STATEMENT OF LINE 7, THEREBY PRODUCING TWO NEW TERMINAL NODES. THE STANDARD INTERPRETER IS CALLED AGAIN TO INTERPRET THE SUBPROBLEM WHICH TOOK THE PATH TO STATEMENT L1 FROM LINE 7. THIS INTERPRETATION STEP INCLUDES AN ATTEMPT TO COMBINE THE FORWARD CONTEXT BEING INTERPRETED WITH THE BACKWARD CONTEXT AT STATEMENT L1. THIS ATTEMPT FAILS BECAUSE THE VALUE OF MONKEY[1] IS X2 IN THE
Forward context and under bananas in the backward context. The interpretation step ends at the computed \texttt{GOTOV} statement of line 9, where three new terminal nodes are produced.

At this point the executive chooses to extend the backward tree since it has three terminal nodes and the forward tree has five. When the inverse interpreter is called to interpret the subproblem which is on the path to the \texttt{GOTOV} statement at line 13 it moves the context back toward statement L1 through lines 12, 11, 10, and 9. After the context is interpreted through the computed \texttt{GOTOV} statement of line 9 an attempt is made to combine it with the forward context also at line 9. The combination attempt is successful and since all the variables in the combined context have values, a solution has been found. The forward, backward, and combined contexts involved in forming this solution are shown in Figure V.14.
Figure V.13. Example two way search for the monkey problem
FORWARD CONTEXT

DATA STRUCTURE

\[ M \]

\[ <M>: 3 \]

BOX

VECTOR: \( X_2, \text{ON. FLOOR} \)

MONKEY

VECTOR: \( X_2, \text{ON. FLOOR} \)

Y

VECTOR: \( \text{ON. FLOOR, ON. BOX} \)

X

VECTOR: \( X_1, X_2, \text{UNDER. BANANAS} \)

VARIABLES

\[ S(2) \]

VALUE: 3

\[ S(1) \]

VALUE: 2

CONSTRAINTS: NONE

BACKWARD CONTEXT

DATA STRUCTURE: EMPTY

VARIABLES

\[ S(99) \]

VALUE: 1

\[ S(98) \]

VALUE: 2

\[ S(97) \]

RANGE: 1 2 3

CONSTRAINTS

\[ X[S(97)] = \text{UNDER. BANANAS} \]

COMBINED CONTEXT

DATA STRUCTURE

\[ M \]

\[ <M>: 3 \]

BOX

VECTOR: \( X_2, \text{ON. FLOOR} \)

MONKEY

VECTOR: \( X_2, \text{ON. FLOOR} \)

Y

VECTOR: \( \text{ON. FLOOR, ON. BOX} \)

X

VECTOR: \( X_1, X_2, \text{UNDER. BANANAS} \)

VARIABLES

\[ S(99) \]

VALUE: 1

\[ S(98) \]

VALUE: 2

\[ S(97) \]

VALUE: 3

\[ S(2) \]

VALUE: 3

\[ S(1) \]

VALUE: 2

CONSTRAINTS: NONE

FIGURE V.14. COMBINATION OF A FORWARD AND A BACKWARD CONTEXT
3.2. MEANS-ENDS ANALYSIS

Given the introduction of the inverse interpreter and the two-way search in ARF, one can propose ways of providing a sense of distance and direction in the subproblem space. Note that when the routine which attempts to combine a forward and a backward context fails, it could output a list of the constraints and data slot values which it was unable to incorporate into the combined context. This output provides a measure of the 'distance' between the two contexts and an indication of what changes need to occur for a successful combination to be possible. Then, instead of the executive interpreting all of the contexts at terminal nodes of the chosen search tree, it could select those contexts which are 'closest' to the contexts at the terminal nodes of the other tree.

A more effective way of using the output from the combination routine would be to analyze the changes needed in the terminal contexts to allow successful combinations. The executive could then attempt to interpret contexts through paths in the ref procedure which would affect the desired changes. For example, in the two-way search for the solution of the monkey problem discussed above, there is an attempt to combine a forward and a backward context at statement L1 which fails. The reason for the


IF THE EXECUTIVE CHOSE TO INTERPRET THE FORWARD CONTEXT INSTEAD OF THE BACKWARD CONTEXT, THEN THE SAME ALTERNATIVES OF INTERPRETING STATEMENT VWALKV OR STATEMENT VMOVE.BOXV WOULD BE AVAILABLE AND A SIMILAR DIFFICULTY WOULD ARISE IN INTERPRETING THROUGH STATEMENT VWALKV.
3.3. AN INITIALIZATION PROBLEM

A basic problem with the use of the inverse interpreter is that the backward contexts do not contain the data slot values defined by the initial statements in a REF procedure. That is, in most cases there are a set of data slot values which are set at the beginning of a REF procedure and are not changed throughout the interpretation of the procedure. During inverse interpretation these are the last values set in a context instead of the first. This situation can cause excessive subproblem proliferation during inverse interpretation. For example, consider inverse interpretation in the crypt-addition problem. Assume the inverse interpreter begins at `VEND` with an empty context, moves through the L2 loop via statement L3, returns to statement L2, and begins interpretation of the statement `VSET <CARRY> TO CARRY AT LINE 18`. The context will contain the constraint `V<A1[2]> + <A2[2]> + <CARRY> = <SUM[2]>` from interpretation of statement L3 the first time through the loop. Note that this constraint could not be further evaluated because the values of the A1, A2, and SUM vectors have not been set. When the inverse interpreter moves the context through the VSET statement at line 18 it must remove all references to <CARRY> in the context. Since it does not know the values in the A1, A2, and SUM vectors it must consider eight cases defined as follows:
## CASE 1
- $A_1[2] = \text{CARRY}$
- $A_2[2] = \text{CARRY}$
- $\text{SUM}[2] = \text{CARRY}$

## CASE 2
- $A_1[2] = \text{CARRY}$
- $A_2[2] = \text{CARRY}$
- $\text{SUM}[2] = \text{CARRY}$

## CASE 3
- $A_1[2] = \text{CARRY}$
- $A_2[2] = \text{CARRY}$
- $\text{SUM}[2] = \text{CARRY}$

## CASE 4
- $A_1[2] = \text{CARRY}$
- $A_2[2] = \text{CARRY}$
- $\text{SUM}[2] = \text{CARRY}$

## CASE 8
- $A_1[2] = \text{CARRY}$
- $A_2[2] = \text{CARRY}$
- $\text{SUM}[2] = \text{CARRY}$

**This case proliferation can negate the advantages of the two way search by introducing unnecessary complexity into the solution process.**

The remedy for this problem is to design an algorithm for analyzing a ref procedure before inverse interpretation begins to determine which data slot values are set once and not changed. These values could then be set initially in each backward context. One cannot hope that this analysis routine could determine for all data slots defined in the procedure whether or not the values change during interpretation, because in general only the actual interpretation can make that determination. One could hope that it would find the most obvious cases of initialization. For example, in the crypt-addition problem the routine should be able to determine that the vectors $A_1$, $A_2$, $\text{SUM}$, and $L$ do not change during interpretation. It is doubtful if the routine could determine that the values assigned to $<D>$, $<N>$, etc., also do not change after being set in the L1 loop.
VI. ARF\textup{VS} \textup{\textbackslash `}SET\textup{\textbackslash `} Statement Interpreter

In this chapter we will consider in detail the task of interpreting a \texttt{\textbackslash `}SET\textup{\textbackslash `} statement in the ARF environment and describe the algorithm used in ARF\textup{\textbackslash `}SET\textup{\textbackslash `} statement interpreter.

The interpretation task can be summarized as follows. A context and a \texttt{\textbackslash `}SET\textup{\textbackslash `} statement are given as input and a list of contexts is to be produced as output. The value defined by the statement's right side expression is to be entered as the value of the data slot (i.e., the vector element or identifier-attribute pair) defined by the statement's left side expression. If the data slot being assigned a value already has a value, then that value will be replaced by the new value. All references to the old value in the context and in the statement's right side expression must be found and replaced by the old value before the new value is entered.

Two situations can occur during this process which necessitate case analysis. The first is that the interpreter may not be able to determine the data slot indicated by the statement's left side. For example consider interpretation of the statement \texttt{\textbackslash `}SET B[<I>]\textup{\textbackslash `} to 3\textup{\textbackslash `} with the following input context:
CONTEXT C0

DATA STRUCTURE

B
  VECTOR: -,7
I
  <I>: S(1)

VARIABLES

S(1)
  VALUE: 2
  RANGE: 1,3,4,5

CONSTRAINTS
  NONE

THE INTERPRETER faces the issue of whether or not to replace the value 7 in the B vector with the value 3. Since the value of S(1) has not yet been determined the interpreter must consider both the case where it is 2 and the 7 is replaced by 3 and the case where it is not 2 and no replacement occurs. Hence, the following two contexts are output at the completion of the interpretation:

CONTEXT C1

DATA STRUCTURE

B
  VECTOR: -,3
I
  <I>: 2

VARIABLES

S(1)
  VALUE: 2

CONSTRAINTS
  NONE

CONTEXT C2

DATA STRUCTURE

B
  VECTOR: -,7
I
  <I>: S(1)

VARIABLES

S(1)
  VALUE: 2
  RANGE: 1,3,4,5

CONSTRAINTS
  B[S(1)] = 3

THE SECOND SITUATION which can cause case analysis to occur is where the interpreter cannot determine if a reference to a data
SLOT IN THE CONTEXT OR IN THE STATEMENT'S RIGHT SIDE REFERS TO THE
SAME FOR EXAMPLE, CONSIDER INTERPRETATION OF THE STATEMENT \texttt{\textbackslash vset B[1] TO 6\textbackslash v} WITH THE INPUT CONTEXT \texttt{C2} FROM THE EXAMPLE ABOVE. THE
INTERPRETER CANNOT DETERMINE IF \texttt{B[S(1)]} IN THE CONTEXT'S
CONSTRAINT IS A REFERENCE TO \texttt{B[1]}. HENCE, BOTH THE CASE WHERE IT
IS AND THE CASE WHERE IT ISN'T MUST BE CONSIDERED SO THAT TWO
CONTEXTS RESULT FROM THE INTERPRETATION AS FOLLOWS:

\begin{tabular}{ll}
\textbf{CONTEXT C2.1} & \textbf{CONTEXT C2.2} \\
\textbf{DATA STRUCTURE} & \textbf{DATA STRUCTURE} \\
\texttt{B} & \texttt{B} \\
\texttt{VECTOR: 6,7} & \texttt{VECTOR: 6,7} \\
\texttt{I} & \texttt{I} \\
\texttt{<I>: 1} & \texttt{<I>: S(1)} \\
\textbf{VARIABLES} & \textbf{VARIABLES} \\
\texttt{S(1)} & \texttt{S(1)} \\
\texttt{VALUE: 1} & \texttt{RANGE: 3,4,5} \\
\textbf{CONSTRAINTS} & \textbf{CONSTRAINTS} \\
\texttt{NONE} & \texttt{B[S(1)]=3} \\
\end{tabular}

NOTE THAT IN EITHER OF THESE TWO SITUATIONS THE CASE ANALYSIS
CAN BE N-ARY RATHER THAN BINARY. FOR EXAMPLE, IF THE B VECTOR HAD
K ELEMENTS IN CONTEXT \texttt{C0}, THEN K+1 CASES COULD BE PRODUCED DURING
THE INTERPRETATION OF THE STATEMENT \texttt{\textbackslash vset B[I] TO 3\textbackslash v}. THE MAXIMUM
NUMBER OF CASES WHICH COULD BE PRODUCED BY THIS INTERPRETATION
WOULD BE LIMITED TO FIVE BY THE RANGE OF \texttt{S(1)}.

THE ALGORITHM USED IN ARP TO CARRY OUT THIS INTERPRETATION IS
FLOW CHARTED IN FIGURES VI.1 AND VI.2. THE ALGORITHM IS DESIGNED
SO THAT REPLACEMENT OF REFERENCES TO THE OLD VALUE AND DELETION OF 
THE OLD VALUE ARE SEPARATED FROM ASSIGNMENT OF THE NEW VALUE. THIS 
SEPARATION ENABLES THE REPLACEMENT AND DELETION PORTION OF THE 
CODE TO BE USED BY ARFV'S INVERSE INTERPRETER DESCRIBED IN CHAPTER 
V.

THE ALGORITHM USES A DUMMY VECTOR TO CARRY OUT THE 
REPLACEMENT AND DELETION AS FOLLOWS. THE PROCESSING PROCEEDS AS IF 
THE OLD VALUE WERE BEING TRANSFERRED TO AN ELEMENT OF VECTOR 
\( \text{\texttt{\textbackslash d\texttt{ummyv}}} \) RATHER THAN BEING DELETED FROM THE CONTEXT, AND ALL 
REFERENCES TO THE VALUE WERE BEING REPLACED BY REFERENCES TO THE 
\( \text{\texttt{\textbackslash d\texttt{ummyv}}} \) VECTOR ELEMENT. HENCE, IN THE EXAMPLE ABOVE WHERE THE 
STATEMENT \( \text{\texttt{\textbackslash vset e\pointless 1 \textless 1\pointless \textgreater \texttt{}} \texttt{3\texttt{}}} \) IS BEING INTERPRETED, THE VALUE 7 IS 
DELETED FROM THE B VECTOR IN CONTEXT C1 AND BECOMES THE VALUE OF 
THE FIRST ELEMENT OF THE \( \text{\texttt{\textbackslash d\texttt{ummyv}}} \) VECTOR. ALSO, DURING THE 
INTERPRETATION OF THE STATEMENT \( \text{\texttt{\textbackslash vset b\pointless 1 \texttt{}} \texttt{6\texttt{}}} \) AS DESCRIBED 
ABOVE, THE CONSTRAINT \( \text{\texttt{\textbackslash v\texttt{b\pointless s\pointless 1\pointless \textgreater \texttt{}} \texttt{3\texttt{}}} \) IN CONTEXT C2.1 BECOMES 
\( \text{\texttt{\textbackslash d\texttt{ummy\pointless 1\pointless \textgreater \texttt{}} \texttt{3\texttt{}}} \). THE \( \text{\texttt{\textbackslash d\texttt{ummyv}}} \) VECTOR IS USED BECAUSE THE OLD VALUE MAY 
NOT OCCUR IN THE CONTEXT'S DATA STRUCTURE (AS IS THE CASE IN 
CONTEXT C2.1 ABOVE) SO THAT THE INTERPRETER CANNOT READILY 
DETERMINE WHAT THE REFERENCES ARE TO BE REPLACED BY.
BEGIN

Evaluate both the statement's left side and right side expressions.

Set evaluated right side as value of S(100).

Increment the context's K value.

Call REPLACER(left side, DUMMY[K value], input context).

Create an empty output list.

Setup to generate the contexts output by REPLACER.

Generate the next context.

Delete the value of S(100) from the generated context.

Add constraint equating statement's left side to the value of S(100).

Add the context to the output list.

Figure VI.1. ARF's SET statement interpreter
Inputs:
- left expression
- right expression
- context

Output:
- context list

BEGIN

Copy the input context and consider the copy the first case.

Setup to generate the data slots in the input context's data structure that match the input left expression.

Generation complete

Generate next matching data slot. A1

Setup to generate the subcases produced from the existing cases by matching the data slot with the input left expression.

A3

Generate the next subcase. Generation complete

Consider the new subcases as the existing cases.

Subcase generated

No

Did the match succeed in this case?

Yes

Delete the value of the matched data slot from the subcase's data structure and add a constraint equating the value to the input right expression.

Figure VI.2A. ARF's replacer subroutine
Figure VI.2B. ARF's replacer subroutine (continued)
Figure VI.20. ARF's replacer subroutine (continued)
ARF contains a cleanup routine which removes irrelevant references to the \[\text{v} \text{DUMMY} \text{v}\] vector and it will remove both the value 7 for \[\text{DUMMY}\{1\}\] in context C1 and the constraint \[\text{v} \text{DUMMY}\{1\} = 3 \text{v}\] in the context C2.1 before the interpreter outputs the contexts.

Figure VI.1 shows the sequence of operations performed by the main routine of the \(\text{v} \text{SETv}\) statement interpreter. The statement\(\text{v}\)'s right side expression is set as the value of pseudo variable \(S\{100\}\) to allow possible references to the data slot indicated by the statement\(\text{v}\)'s left side to be removed from it. The call of the replacer subroutine defines the necessary cases and performs the value deletions and reference replacements described above. The interpreter completes its \(\text{v} \text{SETv}\) statement processing by adding a constraint to each case equating the statement\(\text{v}\)'s altered right side expression (the value of \(S\{100\}\) in each case) to the statement\(\text{v}\)'s left side. This operation effectively sets the new value for the data slot indicated by the statement\(\text{v}\)'s left side in each case.

The replacer routine has three inputs: a left expression, a right expression, and a context. It creates a set of cases (i.e., contexts) and outputs them as a list. Cases are created by matching each data slot in the input context\(\text{v}\)'s data structure with the input left expression. For each case in which a data slot is
SUCCESSFULLY MATCHED, THE VALUE OF THAT DATA SLOT IS DELETED AND A CONSTRAINT IS ADDED TO THE CONTEXT EQUATING THE VALUE TO THE INPUT RIGHT EXPRESSION. FURTHER CASES ARE CREATED FOR EACH SLOT EXPRESSION OCCURRING IN THE INPUT CONTEXT'S VALUES AND CONSTRAINTS WHICH CAN BE MATCHED WITH THE INPUT LEFT EXPRESSION. FOR EACH CASE IN WHICH A SLOT EXPRESSION IS SUCCESSFULLY MATCHED, THE SLOT EXPRESSION IS REPLACED BY THE INPUT RIGHT EXPRESSION. THE CASES THAT ARE PRODUCED DURING A MATCH ARE INDICATED IN Figure VI.3.
STATEMENT FORM 1: SET A1[A2] TO A3;

EXPRESSION BEING MATCHED: A4[A5]

CASE 1: MATCH SUCCESSFUL. ADD CONSTRAINTS A1=A4 AND A2=A5.
CASE 2: MATCH FAILS. ADD CONSTRAINT ~(A1=A4).
CASE 3: MATCH FAILS. ADD CONSTRAINT ~(A2=A5).

STATEMENT FORM 2: SET A1 OF A4 TO A3;

EXPRESSION BEING MATCHED: A6 OF A7

CASE 1: MATCH SUCCESSFUL. ADD CONSTRAINTS A1=A6 AND A4=A7.
CASE 2: MATCH FAILS. ADD CONSTRAINT ~(A1=A6).
CASE 3: MATCH FAILS. ADD CONSTRAINT ~(A4=A7).

STATEMENT FORM 3: SET <A1> TO A3;

EXPRESSION BEING MATCHED: <A4>

CASE 1: MATCH SUCCESSFUL. ADD CONSTRAINT A1=A4.
CASE 2: MATCH FAILS. ADD CONSTRAINT ~(A1=A4).

WHERE A3 IS ANY EXPRESSION; A1, A4, A6, AND A7 ARE SYMBOL EXPRESSIONS; AND A2 AND A5 ARE INTEGER EXPRESSIONS.

FIGURE VI.3. CASES PRODUCED BY THE MATCHING PROCESS

B[2]. At the completion of the generation at A1 we have two cases as follows:

**CONTEXT C1**

DATA STRUCTURE
- B
- VECTOR: -, -

VARIABLES
- S(1)
  - RANGE: 1, 2, 3, 4, 5
  - S(100)
  - VALUE: 3

CONSTRAINTS
- DUMMY[1]=7
- S(1)=2

**CONTEXT C2**

DATA STRUCTURE
- B
- VECTOR: -, 7

VARIABLES
- S(1)
  - RANGE: 1, 2, 3, 4, 5
  - S(100)
  - VALUE: 3

CONSTRAINTS
- S(1) -= 2

There are no slot expressions to generate at A2. The generation at A5 will incorporate the constraints into the contexts so that REPLACERV will output contexts C1 and C2 as follows:

**CONTEXT C1**

DATA STRUCTURE
- B
- VECTOR: -, -
- DUMMY
- VECTOR: 7

VARIABLES
- S(1)
  - RANGE: 1, 3, 4, 5
  - S(100)
  - VALUE: 3

CONSTRAINTS
- NONE

**CONTEXT C2**

DATA STRUCTURE: B
- VECTOR: -, 7

VARIABLES
- S(1)
  - RANGE: 1, 2, 3, 4, 5
  - S(100)
  - VALUE: 3

CONSTRAINTS
- NONE

The equating of B[S(1)] to 3 in each context completes the interpretation of the statement. The resulting contexts are as
FOLLOWS:

<table>
<thead>
<tr>
<th>CONTEXT C1</th>
<th>CONTEXT C2</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA STRUCTURE</td>
<td>DATA STRUCTURE</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>VECTOR: -, 3</td>
<td>VECTOR: -, 7</td>
</tr>
<tr>
<td>DUMMY</td>
<td>VARIABLES</td>
</tr>
<tr>
<td>VECTOR: 7</td>
<td>S(1)</td>
</tr>
<tr>
<td>VARIABLES</td>
<td>RANGE: 1, 3, 4, 5</td>
</tr>
<tr>
<td>S(1)</td>
<td>CONSTRAINTS</td>
</tr>
<tr>
<td>VALUE: 2</td>
<td>B[S(1)]=3</td>
</tr>
<tr>
<td>CONSTRAINTS</td>
<td>NONE</td>
</tr>
</tbody>
</table>

CONTEXT C1

DATA STRUCTURE
B
VECTOR: 6,3
DUMMY
VECTOR: 7

VARIABLES
S(1)
VALUE: 2

CONSTRAINTS
NONE

CONTEXT C2.1

DATA STRUCTURE
B
VECTOR: 6,7
DUMMY
VECTOR: -,3

VARIABLES
S(1)
VALUE: 1

CONSTRAINTS
NONE

CONTEXT C2.2

DATA STRUCTURE
B
VECTOR: 6,7

VARIABLES
S(1)
RANGE: 3,4,5

CONSTRAINTS
B[S(1)]=3

SINCE THE DUMMY VECTOR ELEMENTS DO NOT APPEAR ELSEWHERE IN THE CONTEXTS THEY WILL BE DELETED BY ARFS CLEANUP ROUTINE.
VII. A SAMPLING OF PROBLEMS

This chapter presents a set of example problems which have been stated in REF and presented to ARF for solution. For each problem we will discuss the representation in REF and the behavior of ARF in attempting the solution. For the problems which the program cannot solve in a reasonable amount of processing time we discuss the difficulties which would be encountered based on hand simulations. The chapter is organized in three sections corresponding to the three classes of problems discussed in Chapter II. We begin by commenting that ARF is a 7000 line program written in the IPL-V programming language Newell et al., 1961) and is run on an IBM 360/67 computer (IBM, 1967).

A. BOOLEAN CONSTRAINT SATISFACTION PROBLEMS

1. A SORTING PROBLEM

Consider a simple sorting problem in which five distinct integers given in an input vector are to be sorted in ascending order and transferred to an output vector. A REF representation of this problem is shown in Figure VII.1. The ∇FOR∇ loop in the REF procedure specifies the ordering in the output vector by requiring that each element of the vector be less than the next element.
BEGIN;

SET VECTOR INPUT TO 3,7,9,6,2;

SET VECTOR OUTPUT TO INPUT[SELECT(1,5)], INPUT[SELECT(1,5)],
INPUT[SELECT(1,5)], INPUT[SELECT(1,5)], INPUT[
SELECT(1,5)];

FOR I=4 DO TO L1;

L1: CONDITION OUTPUT[I]<OUTPUT[I]+1;

END;

FIGURE VII. 1. REF STATEMENT OF A SORTING PROBLEM
A REF statement of the problem without the restriction that the integers be distinct is given in Figure VII.2. In this case it is necessary to introduce an intermediate vector to allow the statement of the condition that no element of the input vector can appear more than once in the output vector. In addition, the ordering requirement has been weakened to allow the possibility of equal adjacent elements in the output vector.

Note, that the REF procedures define the problem of sorting an explicit set of integers, rather than defining the more general problem of sorting any given set of integers. If the REF language allowed parameterized subroutines (or procedures), then the more general problem could be defined by denoting the integers to be sorted as input parameters.
BEGIN;

SET_VECTOR INPUT TO 3,7,9,6,3;

SET_VECTOR E TO SELECT(1,5),SELECT(1,5),SELECT(1,5),
SELECT(1,5),SELECT(1,5);

CONDITION EXCL(B[1],B[2],B[3],B[4],B[5]);

SET_VECTOR OUTPUT TO INPUT[B[1]],INPUT[B[2]],INPUT[B[3]],
INPUT[B[4]],INPUT[B[5]];

FOR I=4 DO TO L1;

L1:CONDITION ~(OUTPUT[I]+1)<OUTPUT[I+1]);

END;

FIGURE VII.2. REF STATEMENT OF A SORTING PROBLEM
ARF was given the problem as stated by the REF procedure of Figure VII.1 and found the solution in the following manner. No case analysis is required during interpretation and Figure VII.3 shows the context after being interpreted to \( \text{vendv} \). ARF begins the backtracking search for a solution by assigning the value 1 to \( S(2) \). The choice of \( S(2) \) to receive the first value is arbitrary since each of the constraints has two variables and \( v < v \) as the main operator. With a value assigned to \( S(2) \) the constraint manipulation methods are able to eliminate two of the constraints, determine a value for \( S(1) \), and reduce the range of \( S(3) \) to three elements. At this point the context is as shown in Figure VII.4. The search continues with attempts to set a value for \( S(3) \). After failing with values 2 and 3, the value of \( S(3) \) is successfully set to 4 to produce the context in Figure VII.5. The solution is found when the value 2 is assigned to \( S(4) \) and the constraint manipulation methods deduce that \( S(5) \) must be 3. The solution required 1.3 minutes of processing time.
CONTEXT 14449

DATA STRUCTURE

DUMMY

I

<1>: 4

OUTPUT

VECTOR: INPUT[S(1)], INPUT[S(2)], INPUT[S(3)], INPUT[S(4)], INPUT[S(5)]

INPUT

VECTOR: 3, 7, 9, 6, 2

VARIABLES

S(5)

RANGE: 1 2 3 4 5

S(4)

RANGE: 1 2 3 4 5

S(3)

RANGE: 1 2 3 4 5

S(2)

RANGE: 1 2 3 4 5

S(1)

RANGE: 1 2 3 4 5

CONSTRAINTS

INPUT[S(4)] + INPUT[S(5)] < 0

INPUT[S(3)] + INPUT[S(4)] < 0

INPUT[S(2)] + INPUT[S(3)] < 0

INPUT[S(1)] + INPUT[S(2)] < 0

Figure VII.3. Context at end of interpretation
CONTEXT 15412

DATA STRUCTURE
DUMMY
I
<T>: 5
OUTPUT
VECTOR: 2, 3, INPUT[S(3)], INPUT[S(4)], INPUT[S(5)]
INPUT
VECTOR: 3, 7, 9, 6, 2

VARIABLES
S(5)
RANGE: 1 2 3 4 5
S(4)
RANGE: 1 2 3 4 5
S(3)
RANGE: 2 3 4
S(2)
VALUE: 1
S(1)
VALUE: 5

CONSTRAINTS
INPUT[S(4)] + INPUT[S(5)] < 3
INPUT[S(3)] + INPUT[S(4)] < 3

Figure VII.4. Context after assignment of value to S(2)
CONTEXT 15429

DATA STRUCTURE

DUMMY

<T>

<input>: 5

OUTPUT

VECTOR: 2, 3, 6, INPUT[S(4)], INPUT[S(5)]

INPUT

VECTOR: 3, 7, 9, 6, 2

VARIABLES

S(5)

<range>: 1 2 3 4 5

S(4)

<range>: 2 3

S(3)

<value>: 4

S(2)

<value>: 1

S(1)

<value>: 5

CONSTRAINTS

INPUT[S(4)] + INPUT[S(5)] < 0

Figure VII.5. Context after assignment of value to S(3)
WE SEE IN THE SOLVING OF THIS PROBLEM THE VALUE OF APPLYING THE CONSTRAINT MANIPULATION METHODS DURING THE BACKTRACKING SEARCH. ONLY FIVE VALUE ASSIGNMENTS ARE REQUIRED TO FIND THE SOLUTION, WHEREAS IF THE CONSTRAINT MANIPULATION METHODS WERE NOT APPLIED THIRTY VALUE ASSIGNMENTS WOULD BE NECESSARY.

2. A MAGIC SQUARE PUZZLE

CONSIDER AGAIN THE MAGIC SQUARE PUZZLE INTRODUCED IN CHAPTER II. THE PROBLEM IS ONE OF DETERMINING AN INTEGER IN THE INTERVAL 1 TO 9 FOR EACH ELEMENT OF A 3-BY-3 MATRIX SUCH THAT NO INTEGER OCCURS MORE THAN ONCE IN THE MATRIX AND THE SUM OF THE INTEGERS IN EACH ROW, COLUMN, AND DIAGONAL IS 15. THE REF STATEMENT OF THIS PROBLEM SHOWN IN FIGURE II.5B USES A NINE ELEMENT VECTOR TO REPRESENT THE MATRIX AND A SEQUENCE OF VCONDITIONV STATEMENTS TO EXPRESS THE CONSTRAINTS ON THE VECTOR ELEMENTS.

ARF FOUND A SOLUTION TO THIS PROBLEM IN THE FOLLOWING MANNER. NO CASE ANALYSIS IS REQUIRED DURING INTERPRETATION, AND THE CONSTRAINT MANIPULATION METHODS ELIMINATE VARIABLES AND RANGE ELEMENTS AS EACH VCONDITIONV STATEMENT IS INTERPRETED. FIGURE VII.6 SHOWS THE CONTEXT AT VENDV BEFORE THE BACKTRACKING SEARCH IS BEGUN. NOTE THAT THE CONSTRAINT MANIPULATION METHODS HAVE ELIMINATED SEVEN VARIABLES FROM THE CONSTRAINTS, DETERMINED AN
INTEGER VALUE FOR S(5), DELETED AN ELEMENT FROM EACH OF THE RANGES
OF S(2) AND S(3), AND CREATED EIGHT NEW v<v CONSTRAINTS.

THE BACKTRACKING SEARCH IS BEGUN BY MAKING THE ARBITRARY
SELECTION OF S(3) TO BE ASSIGNED A VALUE. WHEN S(3) IS ASSIGNED
THE VALUE 1 THE CONSTRAINT v10<S(3)+S(2)+S(3)v IMPLIES THAT S(2)
MUST BE 9; BUT THE VALUE OF S(7) IS 10-S(3) AND IS THEREFORE 9
ALSO. THIS CAUSES THE vEXCLv CONSTRAINT TO BECOME FALSE AND
IMPLIED THAT THE VALUE OF S(3) CANNOT BE 1. WHEN S(3) IS ASSIGNED
THE VALUE 2 ALL BUT ONE OF THE CONSTRAINTS REDUCES TO TRUE AND THE
CONTEXT SHOWN IN FIGURE VII.7 IS PRODUCED. NOTE THAT THE RANGE OF
S(2), THE ONLY UNVALUED VARIABLE, HAS BEEN REDUCED TO TWO
ELEMENTS. THE ASSIGNMENT OF 7 AS THE VALUE OF S(2) SUCCESSFULLY
ELIMINATES THE FINAL CONSTRAINT AND PRODUCES THE SOLUTION SHOWN IN
FIGURE VII.8. THE SOLUTION PROCESS REQUIRED 1.9 MINUTES OF
PROCESSING TIME.
DATA STRUCTURE

VECTOR: 15\, +\, -\, S(2)\, +\, -\, S(3)\, ,\, S(2)\, ,\, S(3)\, ,\, -10\, +\, S(3)\, +\, S(2)\, +\, S(3)\, ,\, 5\, ,\, 20\, +\, S(3)\, -\, -\, S(2)\, +\, -\, S(3)\, ,\, 10\, -\, S(3)\, ,\, 10\, +\, -\, S(2)\, ,\, -5\, +\, S(2)\, +\, S(3)

VARIABLES

\begin{align*}
S(9) & \quad \text{VALUE: } -5\, +\, S(2)\, +\, S(3) \\
\text{RANGE: } 1 & \, 2 & \, 3 & \, 4 & \, 5 & \, 6 & \, 7 & \, 8 & \, 9
\end{align*}

\begin{align*}
S(8) & \quad \text{VALUE: } 10\, +\, -\, S(2) \\
\text{RANGE: } 1 & \, 2 & \, 3 & \, 4 & \, 5 & \, 6 & \, 7 & \, 8 & \, 9
\end{align*}

\begin{align*}
S(7) & \quad \text{VALUE: } 10\, +\, -\, S(3) \\
\text{RANGE: } 1 & \, 2 & \, 3 & \, 4 & \, 5 & \, 6 & \, 7 & \, 8 & \, 9
\end{align*}

\begin{align*}
S(6) & \quad \text{VALUE: } 20\, +\, -\, S(3)\, +\, -\, S(2)\, +\, -\, S(3) \\
\text{RANGE: } 1 & \, 2 & \, 3 & \, 4 & \, 5 & \, 6 & \, 7 & \, 8 & \, 9
\end{align*}

\begin{align*}
S(5) & \quad \text{VALUE: } 5 \\
\text{RANGE: } 1 & \, 2 & \, 3 & \, 4 & \, 5 & \, 6 & \, 7 & \, 8 & \, 9
\end{align*}

\begin{align*}
S(4) & \quad \text{VALUE: } -10\, +\, S(3)\, +\, S(2)\, +\, S(3) \\
\text{RANGE: } 1 & \, 2 & \, 3 & \, 4 & \, 5 & \, 6 & \, 7 & \, 8 & \, 9
\end{align*}

\begin{align*}
S(3) & \quad \text{RANGE: } 1 & \, 2 & \, 3 & \, 4 & \, 5 & \, 6 & \, 7 & \, 8 & \, 9
\end{align*}

\begin{align*}
S(2) & \quad \text{RANGE: } 1 & \, 2 & \, 3 & \, 4 & \, 5 & \, 6 & \, 7 & \, 8 & \, 9
\end{align*}

\begin{align*}
S(1) & \quad \text{VALUE: } 15\, +\, -\, S(2)\, +\, -\, S(3) \\
\text{RANGE: } 1 & \, 2 & \, 3 & \, 4 & \, 5 & \, 6 & \, 7 & \, 8 & \, 9
\end{align*}

CONSTRAINTS

\begin{align*}
\text{EXCL} & (15\, +\, -\, S(2)\, +\, -\, S(3)\, ,\, S(2)\, ,\, S(3)\, ,\, -10\, +\, S(3)\, +\, S(2)\, +\, S(3)\, ,\, 5\, ,\, 20\, +\, -\, S(3)\, +\, S(2)\, +\, -\, S(3)\, ,\, 10\, -\, S(3)\, ,\, 10\, +\, -\, S(2)\, ,\, -5\, +\, S(2)\, +\, S(3)) \\
S(2) & \, S(3) < 15 \\
5 & \leq S(2) \, + \, S(3) \\
S(3) & \, S(2) \, + \, S(3) \, < 20 \\
10 & \leq S(3) \, + \, S(2) \, + \, S(3) \\
-\, S(3) & \, -\, S(2) \, + \, -\, S(3) < 10 \\
-20 & \leq S(3) \, + \, -\, S(2) \, + \, -\, S(3) \\
-\, S(2) & \, -\, S(3) < 5 \\
-15 & \leq -\, S(2) \, + \, -\, S(3)
\end{align*}

Figure VII.6. Context at end of interpretation
DATA STRUCTURE

VECTOR: 13+S(2), 3(2), 2, -6+S(2), 5, 16-S(2), 9, 10+S(2), -3+S(2)

VARIABLES

S(9)
VALUE: -3+S(2)
RANGE: 1 2 3 4 5 6 7 8 9

S(9)
VALUE: 10+S(2)
RANGE: 1 2 3 4 5 6 7 8 9

S(7)
VALUE: 8

S(6)
VALUE: 16-S(2)
RANGE: 1 2 3 4 5 6 7 8 9

S(5)
VALUE: 5

S(4)
VALUE: -6+S(2)
RANGE: 1 2 3 4 5 6 7 8 9

S(3)
VALUE: 2

S(2)
RANGE: 7 9

S(1)
VALUE: 13-S(2)
RANGE: 1 2 3 4 5 6 7 8 9

CONSTRAINTS

EXCL(13+S(2), 3(2), 2, -6+S(2), 5, 16-S(2), 3, 10+S(2), -3+S(2))

Figure VII.7. Context after assignment of value to S(3)
DATA STRUCTURE

VECTOR: 6, 7, 2, 1, 5, 9, 9, 3, 4

VARIABLES

S(9) VALUE: 4
S(8) VALUE: 3
S(7) VALUE: 9
S(6) VALUE: 9
S(5) VALUE: 5
S(4) VALUE: 1
S(3) VALUE: 2
S(2) VALUE: 7
S(1) VALUE: 6

CONSTRAINTS: NONE

Figure VII.8. Solution context
WE SEE THAT FOR THIS PROBLEM THE CONSTRAINT MANIPULATION METHODS PLAY THE MAJOR ROLE IN THE SOLUTION PROCESS. IN PARTICULAR, THE CREATION OF \( v < v \) CONSTRAINTS EACH TIME A VARIABLE IS ASSIGNED AN EXPRESSION AS A VALUE SERVES TO MINIMIZE THE EXTENT OF THE BACKTRACKING SEARCH. ONLY THREE ASSIGNMENTS ARE REQUIRED DURING THE SEARCH TO FIND THE SOLUTION, WHEREAS WITHOUT THE \( v < v \) CONSTRAINTS ELEVEN ASSIGNMENTS WOULD BE NECESSARY.

3. THE PICNIC PROBLEM

WE NOW CONSIDER AN EXAMPLE OF A TYPE OF PROBLEM FAMILIAR TO PUZZLE FANS IN WHICH A SITUATION INVOLVING SEVERAL QUANTITIES AND RELATIONSHIPS IS DESCRIBED IN ENGLISH AND THE PROBLEM SOLVER IS ASKED TO DETERMINE SOME UNKNOWN IN THE SITUATION. THE PICNIC PROBLEM APPEARS IN POLYA (1962) AND IS STATED AS FOLLOWS:

**AL, BILL, AND CHRIS PLANNED A BIG PICNIC. EACH BOY SPENT 9 DOLLARS. EACH BOUGHT SANDWICHES, ICE CREAM, AND SODA POP. FOR EACH OF THESE ITEMS THE BOYS SPENT JOINTLY 9 DOLLARS, ALTHOUGH EACH BOY SPLIT HIS MONEY DIFFERENTLY AND NO BOY PAID THE SAME AMOUNT OF MONEY FOR TWO DIFFERENT ITEMS. THE GREATEST SINGLE EXPENSE WAS WHAT AL PAID FOR ICE CREAM; BILL SPENT TWICE AS MUCH FOR SANDWICHES AS FOR ICE CREAM. HOW MUCH DID CHRIS PAY FOR SODA POP. (ALL AMOUNTS ARE IN ROUND DOLLARS.)**

TO STATE THIS PROBLEM IN REP (OR IN ANY FORMAL LANGUAGE) ONE MUST IDENTIFY AND REPRESENT THE QUANTITIES IN THE PROBLEM AND THE
RELATIONS WHICH MUST HOLD AMONG THEM IN A SOLUTION. WE PRESENT A
REP STATEMENT OF THE PROBLEM IN FIGURE VII.9 IN WHICH THE NINE
PURCHASE PRICES ARE REPRESENTED BY A NINE ELEMENT VECTOR vAv AS
FOLLOWS:

|----|------|------|------|

THE vSET.vVECTORv STATEMENT INDICATING THAT EACH OF THE PURCHASE
PRICES ARE TO BE SELECTED FROM THE INTERVAL 1 TO 9 REPRESENTS A
DEPARTURE FROM THE ORIGINAL STATEMENT OF THE PROBLEM WHERE ONLY
FOR THE PURCHASE PRICES IS EXPLICITLY STATED. A MORE FAITHFUL
TRANSLATION OF THE PROBLEM WOULD STATE THAT A VALUE FOR A[9] IS TO
BE SELECTED FROM THE POSITIVE INTEGERS AND THAT EACH OF THE OTHER
ELEMENTS OF THE vAv VECTOR CONTAINS AN UNKNOWN POSITIVE INTEGER;
BUT THE ONLY FACILITY IN REP FOR DEFINING UNKNOWN QUANTITIES IS
THE vSELECTv FUNCTION AND IT REQUIRES THAT A FINITE RANGE BE
GIVEN.
BEGIN;
SET VECTOR A TO SELECT(1,9), SELECT(1,9), SELECT(1,9),
SELECT(1,9), SELECT(1,9), SELECT(1,9);
CONDITION EXCL(A[4], A[5], A[6]);
CONDITION EXCL(A[7], A[8], A[9]);
FOR I = 9 DO TO L1;
IF <I> = 2 THEN L1;
L1:
END;

FIGURE VII.9. REF STATEMENT OF THE PICNIC PROBLEM
The first six \texttt{Condition} statements in the \texttt{Ref} procedure indicate that each boy spent nine dollars and that nine dollars was spent for each type of food. The three \texttt{Condition} statements containing disjunctions indicate that each boy split his money differently, and the three \texttt{Condition} statements containing \texttt{Excl} expressions indicate that no boy paid the same amount for two items. The \texttt{For} loop indicates that the greatest single expense was what Al paid for ice cream (i.e., \texttt{A[2]}). Finally, the \texttt{Condition} statement preceding the \texttt{End} statement indicates that Bill spent twice as much for sandwiches as for ice cream.

\textbf{Arf} was given this \texttt{Ref} statement of the picnic problem and solved it in the following manner. No case analysis is required during interpretation and the constraint manipulation methods eliminate variables and range elements as each \texttt{Condition} statement is interpreted. Figure \ref{fig:vii.10} shows the context at \texttt{End} before the backtracking search is begun. Note that the constraint manipulation methods have eliminated six variables from the constraints, deleted seven of the nine elements in the range of \texttt{S(5)}, and created eight new \texttt{<v} constraints.
DATA STRUCTURE

T

<|T|: 10

A


VARIABLES

S(9)
VALUE: S(5)+S(5)+S(5)+S(3)
RANGE: 1 2 3 4 5 6 7 8 9

S(8)
VALUE: 9+S(5)+S(2)
RANGE: 1 2 3 4 5 6 7 8 9

S(7)
VALUE: -S(5)+S(5)+S(2)+S(3)
RANGE: 1 2 3 4 5 6 7 8 9

S(6)
VALUE: 9+S(5)+S(5)+S(5)
RANGE: 1 2 3 4 5 6 7 8 9

S(5)
RANGE: 1 2

S(4)
VALUE: S(5)+S(5)
RANGE: 1 2 3 4 5 6 7 8 9

S(3)
RANGE: 1 2 3 4 5 6 7 8 9

S(2)
RANGE: 1 2 3 4 5 6 7 8 9

S(1)
VALUE: 9+S(2)+S(3)
RANGE: 1 2 3 4 5 6 7 8 9

CONSTRAINTS

S(5)+S(5)+S(5)+S(3)+S(2)<0
-S(5)+S(5)+S(3)<0
-S(5)+S(5)+S(5)+S(2)<0
S(5)+S(5)+S(2)<0
EXCL (-S(5)+S(5)+S(2)+S(3), 9+S(5)+S(2)+S(3))
-vv (-S(5)+S(2)+S(3)=0) vv (-S(5)+S(2)+S(3)=9)
-S(3)=9
S(5)+S(5)+S(3)+S(3)<10
-S(5)+S(5)+S(2)+S(3)<10
-S(5)+S(5)+S(3)<0
-S(5)+S(3)<0
-S(5)+S(3)<0
EXCL (9+S(2)+S(3), S(2), S(3))
-S(5)+S(2)<1
-9<-S(5)+S(2)
-S(2)+S(3)<1
-9<-S(2)+S(3)

Figure VII.10. Context at end of interpretation
THE BACKTRACKING SEARCH IS BEGUN BY MAKING THE ARBITRARY
SELECTION OF S(5) TO BE ASSIGNED A VALUE. WHEN S(5) IS ASSIGNED
THE VALUE 1 THE CONSTRAINTS $v-9<-S(5)+-S(2)v$ AND $v-S(5)+-S(5)+$
$-S(5)+-S(2)<-9v$ REDUCE TO $vS(2)<8v$ AND $v6<S(2)v$, THEREFORE
IMPLYING THAT THE VALUE OF S(2) MUST BE 7. WITH 7 ASSIGNED AS THE
VALUE OF S(2) THE CONSTRAINT $v-9<-S(2)+-S(3)v$ REDUCES TO $vS(3)<2v$,
THEREFORE IMPLYING THAT THE VALUE OF S(3) MUST BE 1. THIS VALUE
ASSIGNMENT CAUSES THE CONSTRAINT $vEXCL(9+-S(2)+-S(3),S(2),S(3))v$
TO REDUCE TO FALSE (I.E., $vEXCL(1,7,1)v$). THIS CAUSES THE ORIGINAL
ASSIGNMENT OF 1 AS THE VALUE OF S(5) TO FAIL, AND THE BACKTRACKING
SEARCH TO CONTINUE BY ASSIGNING 2 AS THE VALUE OF S(5). THIS
ASSIGNMENT SUCCEEDS AND PRODUCES THE CONTEXT SHOWN IN FIGURE
VII.11. NOTE THAT ALL BUT TWO ELEMENTS HAVE BEEN DELETED FROM THE
RANGE OF S(2) AND ONLY THREE ELEMENTS REMAIN IN THE RANGE OF S(3).
CONTEXT 449205

DATA STRUCTURE

DUMMY

I

A

VECTORS: 9-S(2)-S(3), S(2), S(3), 4, 2, 3, -4+S(2)+S(3), 7-S(2), 6-S(3)

VARIABLES

S(9)
VALUE: 6+-S(3)
RANGE: 1 2 3 4 5 6 7 8 9
S(9)
VALUE: 7+-S(2)
RANGE: 1 2 3 4 5 6 7 8 9
S(7)
VALUE: -4+S(2)+S(3)
RANGE: 1 2 3 4 5 6 7 8 9
S(6)
S(5)
VALUE: 3
S(4)
VALUE: 2
S(3)
VALUE: 4
S(2)
RANGE: 1 2 3
S(2)
RANGE: 5 6
S(1)
VALUE: 9+-S(2)+-S(3)
RANGE: 1 2 3 4 5 6 7 8 9

CONSTRAINTS

-S(3)+-S(2)<-6
EXCL(-4+S(2)+S(3), 7+-S(2), 6+-S(3))
=S(2)+S(3)=8) v=(S(2)=5) v=(S(3)=3)
=(S(2)+S(3)=5) v=(S(2)=2) v=(S(3)=3)
S(2)+S(3)<14
4<=S(2)+S(3)
S(3)+-S(2)<0
-S(2)+-S(3)+-S(2)<-9
EXCL(9+-S(2)+-S(3), S(2), S(3))
-S(2)+-S(3)<1
-9<-S(2)+-S(3)

Figure VII.11. Context after assignment of value to S(5)
The search proceeds by assigning 5 as the value of $s(2)$. This assignment causes the constraints $
eg(s(2)+s(3)=8)$ and $
eg(s(2)=5)$ and $(s(3)=3)$ to reduce to $\neg(s(3)=3)$ and $s(3)<6$, therefore implying that the value of $s(3)$ must be 2. When $s(3)$ is assigned the value 2 the constraint $\neg\text{excl}(9+s(2)+s(3),s(2),s(3))$ reduces to false (i.e., $\neg\text{excl}(2,7,2)$). This failure causes $s(2)$ to be assigned the value 6. This assignment implies the value 1 for $s(3)$ and produces the solution shown in Figure VII.12. The solution process required 4.7 minutes.
\textbf{CONTEXT 449046}

\textbf{DATA STRUCTURE}
\textbf{DUMMY}
\begin{itemize}
  \item \textbf{I}
    \begin{itemize}
      \item \textbf{<I>}: 10
    \end{itemize}
  \end{itemize}
\begin{itemize}
  \item \textbf{A VECTOR}: 2, 6, 1, 4, 2, 3, 3, 1, 5
\end{itemize}

\textbf{VARIABLES}
\begin{itemize}
  \item \textbf{S(9) VALUE}: 5
  \item \textbf{S(8) VALUE}: 1
  \item \textbf{S(7) VALUE}: 3
  \item \textbf{S(6) VALUE}: 3
  \item \textbf{S(5) VALUE}: 2
  \item \textbf{S(4) VALUE}: 4
  \item \textbf{S(3) VALUE}: 1
  \item \textbf{S(2) VALUE}: 6
  \item \textbf{S(1) VALUE}: 2
\end{itemize}

\textbf{CONSTRAINTS: NONE}

\textbf{Figure VII.12. Solution context}
ONE OF ARF'S DIFFICULTIES IN SOLVING THIS PROBLEM IS THAT IT MUST PROCESS THE CONSTRAINTS IN THE ORDER THAT THEY ARE CREATED BY THE INTERPRETER. TO TEST THE MAGNITUDE OF THIS DIFFICULTY WE GAVE THE PROBLEM TO A MODIFIED VERSION OF ARF IN WHICH NO CONSTRAINT MANIPULATION METHODS ARE APPLIED UNTIL A CONTEXT IS INTERPRETED TO \texttt{vend\textasciitilde}. DURING INTERPRETATION THE ONLY PROCESSING DONE ON NEW CONSTRAINTS IS EVALUATION AND TESTS FOR A VALUE OF \texttt{true\textasciitilde} OR \texttt{false\textasciitilde}. ONCE THE CONTEXT IS INTERPRETED TO \texttt{vend\textasciitilde} THE SELECTION ROUTINE DESCRIBED IN CHAPTER IV IS USED TO DETERMINE THE ORDER IN WHICH THE CONSTRAINTS ARE PROCESSED.

THIS MODIFIED ARF INTERPRETED THE CONTEXT FOR THE PICNIC PROBLEM TO \texttt{vend\textasciitilde} IN 0.4 MINUTES BUT 3.6 MINUTES WAS REQUIRED BEFORE ALL THE CONSTRAINTS WERE PROCESSED. THIS COMPARES WITH 4.1 MINUTES FOR THE UNMODIFIED ARF TO COMPLETE THE INTERPRETATION AND CONSTRAINT PROCESSING. THE REDUCTION OF LESS THAN 13 PERCENT IN PROCESSING TIME INDICATES THAT CONSIDERING CONSTRAINTS IN AN OPTIMAL ORDER ONLY PROVIDES A LARGE REDUCTION IN PROCESSING TIME WHEN A CONTRADICTION IS FOUND DURING THE PROCESSING. THAT IS, IF ALL THE CONSTRAINTS MUST BE PROCESSED THEN THE ORDER IN WHICH THE PROCESSING IS DONE DOES NOT SIGNIFICANTLY AFFECT THE REQUIRED PROCESSING TIME.
4. THE CONFUSION OF PATENTS PROBLEM

WE NOW CONSIDER ANOTHER EXAMPLE OF A PUZZLE IN WHICH A SITUATION IS DESCRIBED IN ENGLISH AND THE PROBLEM SOLVER IS ASKED TO DETERMINE SOME UNKNOWN RELATIONSHIPS IN THE SITUATION. THIS PROBLEM, STATED IN FIGURE VII.13, INVOLVES FIVE INVENTORS WHO LIVE IN FIVE DIFFERENT CITIES AND WHO HAVE THE SAME PATENT ATTORNEY. THE ATTORNEY SENDS OUT A SET OF PATENTS TO THE INVENTORS BUT DUE TO AN ERROR EACH PATENT IS SENT TO THE WRONG PERSON. THE PROBLEM IS TO DETERMINE WHO SHOULD HAVE RECEIVED WHAT WHERE. THE SOLUTION IS CONSTRAINED BY A SET OF STATEMENTS WHICH PROVIDE FURTHER INFORMATION ABOUT THE INVENTORS, INVENTIONS, CITIES, AND PATENTS.
A certain patent attorney was astonished when he received the simultaneous allowance of five patents, for five separate clients, each of whom lived in a different city.

His astonishment turned to chagrin, however, when he learned what had happened to the patents. They had been received in his office on the same day, but due to an error of a new clerk were sent out in wrong envelopes. Each client received a patent -- but not his own.

The inventor of the steam-shovel received the mouse-trap patent, while the inventor of the latter found in his mail the papers which should have gone to Mr. Green. Mr. Blue received the patent for the rumble-seat awning. Mr. Black's patent was sent to Chicago; the patent which should have gone there was sent to Boston.

Mr. Brown had the patent intended for New York. Mr. White had Mr. Brown's patent. The non-refillable bottle patent was sent to Los Angeles. The inventor of the bottle received the patent of the Cleveland client, while in Cleveland the surprised client received a patent for an anti-snore device.

Who should have received what where.

Figure VII.13. English statement of the confusion of patents problem.
THIS PROBLEM IS A DIFFICULT ONE FOR HUMANS TO SOLVE. THE PRIMARY DIFFICULTY IS IN DETERMINING HOW TO MAKE DEDUCTIONS FROM THE INFORMATION GIVEN. THE PROBLEM PRESENTS A SET OF HIGHLY INTERRELATED ELEMENTS, NAMELY THE INVENTORS, THE INVENTIONS, THE CITIES, AND THE PATENTS. THE STATEMENTS IN THE PROBLEM REQUIRE THE SOLVER TO BE AWARE OF THESE RELATIONSHIPS AND TO MAKE USE OF THEM IN DETERMINING A SOLUTION.

IN FIGURE VII.14A AND VII.14B WE PRESENT TWO FORMULATIONS OF THIS PROBLEM IN REF. BOTH FORMULATIONS USE THE SAME REPRESENTATION FOR THE ELEMENTS IN THE PROBLEM AND FOR CONSTRAINTS STATED IN PARAGRAPHS 3 AND 4 OF THE ORIGINAL PROBLEM STATEMENT. THEY DIFFER IN THEIR MANNER OF STATING THAT EACH DEVICE WAS INVENTED BY A DIFFERENT PERSON, THAT EACH INVENTOR LIVES IN A DIFFERENT CITY, AND THAT EACH PATENT WAS SENT TO THE WRONG PERSON.
BEGIN;

SET VECTOR INVENTION TO STEAM.SHOVEL,MOUSE.TRAP,
RUMBLESEAT.AWNING,NONREFILLABLE.BOTTLE,ANTISNORE.DEVICE;
SET VECTOR CITY TO CHICAGO,BOSTON,NEW.YORK,LOS.ANGELES,
CLEVELAND;
SET VECTOR INVENTOR TO GREEN,BLUE,BLACK,BROWN,WHITE;
FOR I=1 TO 5 DO TO L1;
SET <CITY> TO CITY[SELECT(1,5)];
SET <INVENTOR> TO INVENTOR[SELECT(1,5)];
SET CITY OF INVENTION[<I>] TO <CITY>;
SET INVENTOR OF INVENTION[<I>] TO <INVENTOR>;
SET <PATENT.SENT> TO INVENTION[SELECT(1,5)];
CONDITION -(<PATENT.SENT>=INVENTION[<I>]);
SET PATENT.SENT OF <INVENTOR> TO <PATENT.SENT>;};
L1: SET PATENT.SENT OF <INVENTOR> TO <PATENT.SENT>;
CONDITION EXCL(CITY OF STEAM.SHOVEL,CITY OF MOUSE.TRAP,
CITY OF RUMBLESEAT.AWNING,CITY OF NONREFILLABLE.BOTTLE,
CITY OF ANTISNORE.DEVICE);
CONDITION EXCL(INVENTOR OF STEAM.SHOVEL,INVENTOR OF
MOUSE.TRAP, INVENTOR OF RUMBLESEAT.AWNING, INVENTOR OF
NONREFILLABLE.BOTTLE, INVENTOR OF ANTISNORE.DEVICE);
CONDITION EXCL(PATENT.SENT OF CHICAGO, PATENT.SENT OF
BOSTON, PATENT.SENT OF NEW.YORK, PATENT.SENT OF
LOS.ANGELES, PATENT.SENT OF CLEVELAND);
CONDITION PATENT.SENT OF INVENTOR OF STEAM.SHOVEL =
MOUSE.TRAP;
CONDITION INVENTOR OF PATENT.SENT OF INVENTOR OF
MOUSE.TRAP = GREEN;
CONDITION PATENT.SENT OF BLUE = RUMBLESEAT.AWNING;
CONDITION INVENTOR OF PATENT.SENT OF CHICAGO = BLACK;
CONDITION CITY OF PATENT.SENT OF BOSTON = CHICAGO;
CONDITION CITY OF PATENT.SENT OF BOSTON = NEW.YORK;
CONDITION INVENTOR OF PATENT.SENT OF WHITE = BROWN;
CONDITION PATENT.SENT OF LOS.ANGELES =
NONREFILLABLE.BOTTLE;
CONDITION CITY OF PATENT.SENT OF INVENTOR OF
NONREFILLABLE.BOTTLE = CLEVELAND;
CONDITION PATENT.SENT OF CLEVELAND = ANTISNORE.DEVICE;
END;

FIGURE VII.14A. FIRST REF FORMULATION OF THE CONFUSION OF PATENTS
PROBLEM
BEGIN;
SET VECTOR INVENTION TO STEAM.SHOVEL,MICE.TRAP,
RUMBLESEAT.AWNING,NONREFILLABLE.BOTTLE,ANTISNORE.DEVICE;
SET VECTOR PATENT.SENT TO INVENTION[SELECT(1,5)],
invention[SELECT(1,5)],INVENTION[SELECT(1,5)],
INVENTION[SELECT(1,5)],INVENTION[SELECT(1,5)];
CONDITION EXCL(PATENT.SENT[1],PATENT.SENT[2],PATENT.SENT
[3],PATENT.SENT[4],PATENT.SENT(5));
SET VECTOR TEMP1 TO CHICAGO,BOSTON,NEW.YORK,LOS.ANGIE,
CLEVELAND;
SET VECTOR CITY TO TEMP1[SELECT(1,5)],TEMP1[SELECT(1,5)],
TEMP1[SELECT(1,5)],TEMP1[SELECT(1,5)],TEMP1[SELECT(1,5)];
CONDITION EXCL(CITY[1],CITY[2],CITY[3],CITY[4],CITY[5]);
SET VECTOR TEMP2 TO GREEN,BLUE,BLACK,BROWN,WHITE;
SET VECTOR INVENTOR TO TEMP2[SELECT(1,5)],TEMP2[SELECT
(1,5)],TEMP2[SELECT(1,5)],TEMP2[SELECT(1,5)],
TEMP2[SELECT(1,5)];
CONDITION EXCL(INVENTOR[1],INVENTOR[2],INVENTOR[3],
INVENTOR[4],INVENTOR(5));
FOR I=5 DO TO L1;
SET CITY OF INVENTION[<I>] TO CITY[<I>];
SET INVENTOR OF INVENTION[<I>] TO INVENTOR[<I>];
SET PATENT.SENT OF CITY[<I>] TO PATENT.SENT[<I>];
SET PATENT.SENT OF INVENTOR[<I>] TO PATENT.SENT[<I>];
L1:CONDITION ~(PATENT.SENT[<I>]=INVENTION[<I>]);
CONDITION PATENT.SENT OF INVENTOR OF STEAM.SHOVEL =
MOUSE.TRAP;
CONDITION INVENTOR OF PATENT.SENT OF INVENTOR OF
MOUSE.TRAP = GREEN;
CONDITION PATENT.SENT OF BLUE = RUMBLESEAT.AWNING;
CONDITION INVENTOR OF PATENT.SENT OF CHICAGO = BLACK;
CONDITION CITY OF PATENT.SENT OF BOSTON = CHICAGO;
CONDITION CITY OF PATENT.SENT OF BROWN = NEW.YORK;
CONDITION INVENTOR OF PATENT.SENT OF WHITE = BROWN;
CONDITION PATENT.SENT OF LOS.ANGIE =
NONREFILLABLE.BOTTLE;
CONDITION CITY OF PATENT.SENT OF INVENTOR OF
NONREFILLABLE.BOTTLE = CLEVELAND;
CONDITION PATENT.SENT OF CLEVELAND = ANTISNORE.DEVICE;
END;

FIGURE VII.14B. SECOND RE? FORMULATION OF THE CONFUSION OF PATENTS
PROBLEM
In both formulations each invention, inventor, and city is represented by an identifier. `select` function calls are used to associate with each invention the city in which it was invented and its inventor, to associate with each city the invention whose patent was sent to the city, and to associate with each inventor the invention whose patent was sent to the inventor. These associations do not express all the inter-relationships among the problem elements, but they are sufficient to express the constraints given in paragraphs 3 and 4 of the problem statement.

In the `for` loop of the first `ref` formulation the selections are made to establish the required associations and the constraint is stated that the patents were all sent to the wrong people. Each time through the loop a city, an inventor, and a patent are selected and the value of `<i>` indicates an invention. Appropriate associations are established in the loop to indicate that the selected inventor invented the indicated invention in the selected city and that he received the selected patent.

The first three `condition` statements following the `for` loop in the first formulation state that each device was invented in a different city, that each device was invented by a different person, and that each patent was sent to a different city. The remaining `condition` statements, which are found in both
FORMULATIONS, ARE TRANSLITERATIONS OF THE STATEMENTS IN PARAGRAPHS 3 AND 4 OF THE ORIGINAL PROBLEM STATEMENT.

ARF WOULD NOT BE ABLE TO SOLVE THIS PROBLEM GIVEN THE FIRST FORMULATION BECAUSE OF THE LARGE NUMBER OF CASES PRODUCED DURING INTERPRETATION. NEW CASES ARE PRODUCED DURING THE INTERPRETATION OF THE LAST TWO STATEMENTS IN THE VFORV LOOP BECAUSE THE VSETV STATEMENT INTERPRETER CANNOT IDENTIFY WHICH ATTRIBUTE-IDENTIFIER PAIR IS BEING ASSIGNED A VALUE BY THOSE STATEMENTS, AND IT DOES NOT KNOW THAT THE VALUES OF <CITY> AND <INVENTOR> MUST BE DIFFERENT EACH TIME THROUGH THE LOOP. AT THE COMPLETION OF THE FIFTH TIME THROUGH THE LOOP, 104 CASES WILL HAVE BEEN PRODUCED. ALL BUT ONE OF THESE CASES WILL BE ELIMINATED BY THE THREE VCONDITIONV STATEMENTS FOLLOWING THE LOOP, BUT ARF WILL PRODUCE ALL 104 CASES BEFORE BEGINNING THE ELIMINATIONS.

THE SECOND FORMULATION OF THIS PROBLEM WAS DESIGNED TO PREVENT THE CASE EXPLOSION PROBLEM. IN IT ALL THE SELECTIONS ARE MADE AND THE NECESSARY VEXCLV CONSTRAINTS ARE ENCOUNTERED BEFORE THE ASSOCIATIONS ARE ENTERED INTO THE CONTEXT BY A VFORV LOOP. THE ORDER OF THE SELECTIONS IS SUCH THAT FOR I=1,...,5 THE ITH SELECTED INVENTOR INVENTED THE ITH ELEMENT OF THE VINVENTIONV VECTOR IN THE ITH SELECTED CITY AND HE RECEIVED THE ITH SELECTED PATENT. ARF CAN INTERPRET THIS PROCEDURE WITHOUT DOING CASE
ANALYSIS BECAUSE THE CONTEXT ALREADY CONTAINS THE \texttt{\textasciitilde EXCL\texttilde} CONSTRAINTS NECESSARY FOR ELIMINATING THE EXTRANEOUS CASES WHEN THE \texttt{\textasciitilde FOR\texttilde} LOOP IS INTERPRETED.

ARF FOUND A SOLUTION TO THIS PROBLEM STATED IN THE SECOND FORMULATION IN THE FOLLOWING MANNER. THE CONTEXT HAS THE FORM SHOWN IN FIGURE VII.15 AFTER BEING INTERPRETED THROUGH THE \texttt{\textasciitilde FOR\texttilde} LOOP. INTERPRETATION OF THE \texttt{\textasciitilde SET\texttilde} STATEMENTS AT LINES 24 AND 25 OF THE PROCEDURE PRODUCED TEN EQUALITY CONSTRAINTS, AND INTERPRETATION OF THE \texttt{\textasciitilde CONDITION\texttilde} STATEMENT L1 REDUCED THE RANGES OF VARIABLES \textit{S}(1) THROUGH \textit{S}(5) TO FOUR ELEMENTS.

WHEN INTERPRETATION REACHES \texttt{\textasciitilde END\texttilde} THE CONTEXT APPEARS AS IN FIGURE VII.16. THE VALUE 2 WAS DETERMINED FOR \textit{S}(1) DURING THE INTERPRETATION OF THE STATEMENT \texttt{\textasciitilde CONDITION\texttilde} STATEMENT \textit{PATENT} \texttt{\textasciitilde S(1)\texttilde} OF INVENTOR OF STEAM\textit{SHOVEL = MOUSE\textit{TRAP}}. THE VALUE 2 IS DEDUCED FROM THE VALUE \texttt{\textasciitilde TEMP2(S(1))\texttilde} FOR \texttt{\textasciitilde INVENTOR\texttilde} OF STEAM\textit{SHOVEL} IN THE CONTEXT'S DATA STRUCTURE AND THE CONSTRAINT \texttt{\textasciitilde PATENT\texttilde} \texttt{\textasciitilde S(1)\texttilde} OF \texttt{\textasciitilde TEMP2(S(1)) = INVENTION(S(1))\texttilde}.
DATA STRUCTURE

ANTISENSE DEVICE
INVENTOR OF ANTI SENSE DEVICE: TEMP(8(15))
CITY OF INVENTOR DEVICE: TEMP(8(16))

ANTISENSE BOTTLE
INVENTOR OF NON-PERISHABLE BOTTLE: TEMP(8(14))
CITY OF INVENTOR BOTTLE: TEMP(8(15))

ANTISENSE SEAT AWNING
INVENTOR OF NON-PERISHABLE SEAT, AWNING: TEMP(8(13))
CITY OF INVENTOR SEAT, AWNING: TEMP(8(14))

ANTISENSE TRAP
INVENTOR OF MOUSE TRAP: TEMP(8(12))
CITY OF MOUSE TRAP: TEMP(8(7))

STEAM SHOVEL
INVENTOR OF STEAM SHOVEL: TEMP(8(11))
CITY OF STEAM SHOVEL: TEMP(8(16))

VARIABLES

INDEX
NAME: 1 2 3 4 5
NAME: 1 2 3 4 5
NAME: 1 2 3 4 5
NAME: 1 2 3 4 5
NAME: 1 2 3 4 5
NAME: 1 2 3 4 5
NAME: 1 2 3 4 5
NAME: 1 2 3 4 5
NAME: 1 2 3 4 5
NAME: 1 2 3 4 5

Figure VII.15. Context after interpretation through FOR loop
CLASSICALS

\( \text{PARENT}. \text{SENT TO} \ \text{TEMP}(S(18)) = \text{INVENTION}(S(5)) \)
\( \text{PARENT}. \text{SENT TO} \ \text{TEMP}(S(19)) = \text{INVENTION}(S(8)) \)
\( \text{PARENT}. \text{SENT TO} \ \text{TEMP}(S(16)) = \text{INVENTION}(S(4)) \)
\( \text{PARENT}. \text{SENT TO} \ \text{TEMP}(S(11)) = \text{INVENTION}(S(6)) \)
\( \text{PARENT}. \text{SENT TO} \ \text{TEMP}(S(12)) = \text{INVENTION}(S(3)) \)
\( \text{PARENT}. \text{SENT TO} \ \text{TEMP}(S(10)) = \text{INVENTION}(S(12)) \)
\( \text{PARENT}. \text{SENT TO} \ \text{TEMP}(S(7)) = \text{INVENTION}(S(1)) \)
\( \text{PARENT}. \text{SENT TO} \ \text{TEMP}(S(11)) = \text{INVENTION}(S(11)) \)
\( \text{PARENT}. \text{SENT TO} \ \text{TEMP}(S(6)) = \text{INVENTION}(S(11)) \)
\( \text{CYCLE}(\text{TEMP}(S(11)), \text{TEMP}(S(12)), \text{TEMP}(S(12)), \text{TEMP}(S(14)), \text{TEMP}(S(15))) \)
\( \text{CYCLE}(\text{TEMP}(S(11)), \text{TEMP}(S(12)), \text{TEMP}(S(3)), \text{TEMP}(S(3)), \text{TEMP}(S(11))) \)
\( \text{CYCLE}(\text{INVENTION}(S(1)), \text{INVENTION}(S(2)), \text{INVENTION}(S(3)), \text{INVENTION}(S(4)), \text{INVENTION}(S(5))) \)

\text{Figure VII.15 (continued)}
Figure VII.16. Context at end of interpretation
\text{VALUE: 7}

\text{CONSTRAINTS}
\begin{align*}
\text{CITY OF PATENT, SENT OF TEMPO(12)} &= \text{CLEVELAND} \\
\text{INVENTOR OF PATENT, SENT OF WHITE} &= \text{GREEN} \\
\text{CITY OF PATENT, SENT OF BROWN} &= \text{NEW YORK} \\
\text{CITY OF PATENT, SENT OF BOSTON} &= \text{CHICAGO} \\
\text{INVENTOR OF PATENT, SENT OF OURCASH} &= \text{BLACK} \\
\text{INVENTOR OF PATENT, SENT OF TEMPO(12)} &= \text{GREEN} \\
\text{INVENTOR, SENT OF TEMPO(12)} &= \text{MOUSE, TRAP} \\
\text{INVENTION(S(2)), INVENTION(S(4))} &\quad \text{INVENTION(S(4))} \\
\text{PATENT, SENT OF TEMPO(12)} &= \text{MOUSE, TRAP} \\
\text{PATENT, SENT OF TEMPO(12)} &= \text{INVENTION(S(4))} \\
\text{PATENT, SENT OF TEMPO(12)} &= \text{INVENTION(S(4))} \\
\text{PATENT, SENT OF TEMPO(12)} &= \text{INVENTION(S(4))} \\
\text{PATENT, SENT OF TEMPO(12)} &= \text{INVENTION(S(4))} \\
\text{PATENT, SENT OF TEMPO(12)} &= \text{INVENTION(S(4))} \\
\text{PATENT, SENT OF TEMPO(12)} &= \text{INVENTION(S(4))} \\
\text{PATENT, SENT OF TEMPO(12)} &= \text{INVENTION(S(4))} \\
\text{EVEN(12), TEMPO(13), TEMPO(14), TEMPO(15)} &\quad \text{INVENTION(S(4))} \\
\text{INVENTION(S(2)), TEMPO(13), TEMPO(14), TEMPO(15)} &\quad \text{INVENTION(S(4))} \\
\text{INVENTION(S(2)), TEMPO(13), TEMPO(14), TEMPO(15)} &\quad \text{INVENTION(S(4))} \\
\text{INVENTION(S(2)), TEMPO(13), TEMPO(14), TEMPO(15)} &\quad \text{INVENTION(S(4))} \\
\end{align*}
NOTE THAT THERE ARE CONSTRAINTS IN THE CONTEXT WHICH CONTAIN NO VARIABLES. EVALUATION OF THESE CONSTRAINTS DEPENDS ON THE VALUES OF THE VARIABLES IN THE CONSTRAINTS PRODUCED DURING THE INTERPRETATION OF THE \texttt{SET} STATEMENTS AT LINES 24 AND 25 OF THE \texttt{REP} PROCEDURE.

THE BACKTRACKING SEARCH PROCEEDS DIRECTLY TO THE SOLUTION WITH ONLY ONE BACKTRACKING STEP NECESSARY. THE ORDER OF THE VALUE ASSIGNMENTS IS AS FOLLOWS: \texttt{S(11)=1, S(6)=1, S(2)=1, S(7)=2, S(14)=2, S(13)=4, AND S(13)=5}. THE ASSIGNMENT OF 4 TO \texttt{S(13)} IS THE ONLY UNSUCCESSFUL ASSIGNMENT. THE DIRECTNESS WITH WHICH THE SOLUTION IS FOUND INDICATES THE EFFECTIVENESS OF THE CONSTRAINT MANIPULATION METHODS FOR DELETING RANGE ELEMENTS WHICH FALSELY CONSTRAINTS. THE TIME REQUIRED FOR FINDING THE SOLUTION WAS 4.5 MINUTES. THE SOLUTION IS SHOWN IN FIGURE VII.17.
DATA STRUCTURE

BROWN
PATENT.SENT OF BROWN: NONREFILLABLE.BOTTLE
WHITE
PATENT.SENT OF WHITE: ANTISNORE.DEVICE
NEW YORK
PATENT.SENT OF NEW YORK: RUMBLESEAT.AWNING
BOSTON
PATENT.SENT OF BOSTON: STEAM.SHOVEL
BLACK
PATENT.SENT OF BLACK: STFAM.SHOVEL
CHICAGO
PATENT.SENT OF CHICAGO: MOUSE.TRAP
GREEN
PATENT.SENT OF GREEN: MOUSE.TRAP
CLEVELAND
PATENT.SENT OF CLEVELAND: ANTISNORE.DEVICE
LOS ANGELES
PATENT.SENT OF LOS ANGELES: NONREFILLABLE.BOTTLE
BLUE
PATENT.SENT OF BLUE: RUMBLESEAT.AWNING
ANTISNORE.DEVICE
INVENTOR OF ANTISNORE.DEVICE: BROWN
CITY OF ANTISNORE.DEVICE: LOS ANGELES
NONREFILLABLE.BOTTLE
INVENTOR OF NONREFILLABLE.BOTTLE: BLUE
CITY OF NONREFILLABLE.BOTTLE: NEW YORK
RUMBLESEAT.AWNING
INVENTOR OF RUMBLESEAT.AWNING: WHITE
CITY OF RUMBLESEAT.AWNING: CLEVELAND
MOUSE.TRAP
INVENTOR OF MOUSE.TRAP: BLACK
CITY OF MOUSE.TRAP: BOSTON
DUMMY
STEAM.SHOVEL
INVENTOR OF STEAM.SHOVEL: GREEN
CITY OF STEAM.SHOVEL: CHICAGO

1
<1>: 6
INVENTOR
VECTOR: GREEN, BLACK, WHITE, BLUE, BROWN
TEMP2
VECTOR: GREEN, BLUE, BLACK, BROWN, WHITE
CITY
VECTOR: CHICAGO, BOSTON, CLEVELAND, NEW YORK, LOS ANGELES
TEMP1
VECTOR: CHICAGO, BOSTON, NEW YORK, LOS ANGELES, CLEVELAND
PARENT.SENT
VECTOR: MOUSE.TRAP, STEAM.SHOVEL, ANTISNORE.DEVICE, RUMBLESEAT.AWNING, NONREFILLABLE.BOTTLE
INVENTION
VECTOR: STEAM.SHOVEL, MOUSE.TRAP, RUMBLESEAT.AWNING, NONREFILLABLE.BOTTLE, ANTISNORE.DEVICE

Figure VII.17. Solution context
VARIABLES
S(15)
  VALUE: 4
S(14)
  VALUE: 2
S(13)
  VALUE: 5
S(12)
  VALUE: 3
S(11)
  VALUE: 1
S(10)
  VALUE: 4
S(9)
  VALUE: 3
S(8)
  VALUE: 5
S(7)
  VALUE: 2
S(6)
  VALUE: 1
S(5)
  VALUE: 4
S(4)
  VALUE: 3
S(3)
  VALUE: 5
S(2)
  VALUE: 1
S(1)
  VALUE: 2

CONSTRAINTS: NONE

Figure VII.17 (continued)
5. THE EIGHT QUEENS PROBLEM

One of the classical problems connected with a chess-board is the determination of the number of ways in which eight queens can be placed on a chess-board so that no queen can take any other (Ball, 1960). The same problem may be stated out of the context of chess as one of selecting eight squares on an 8-by-8 array of squares so that no more than one square is selected in each row, column, and diagonal of the array.

We present a rep statement of this problem in Figure VII.18 which is derived from a formulation of the problem by Floyd (1967). The rep procedure indicates placement of a queen on each of the 8 rows by the selection of a column number for each row; i.e., the selection of \( J \) for the value of \( R[I] \) indicates placement of a queen on column \( J \) of row \( I \). The procedure's first condition statement indicates that no column can have more than one queen. The other two condition statements indicate that no diagonal can have more than one queen. The manner in which the selections are made guarantees that each row will have exactly one queen.
BEGIN;

SET_VECTOR R TO SELECT(1,8),SELECT(1,8),SELECT(1,8), SELECT(1,8), SELECT(1,8), SELECT(1,8), SELECT(1,8), SELECT(1,8);

CONDITION EXCL(R[1],R[2],R[3],R[4],R[5],R[6],R[7],R[8]);


CONDITION EXCL(R[1]-1,R[2]-2,R[3]-3,R[4]-4,R[5]-5,R[6]-6,R[7]-7,R[8]-8);

END;

FIGURE VII.18. REFSTATEMENT OF THE 8 QUEENS PROBLEM
ARF solved this problem in the following manner. No case analysis was necessary during interpretation and no deductions were made by the constraint manipulation methods before the backtracking search began. During the search the phase 2 processor for *vexclv* constraints (see Chapter IV, Section 3.2) played the primary role in reducing the amount of search required to find the solution.

Figure VII.19 shows the first 12 steps in the search and illustrates the effect of the phase 2 *vexclv* processor. Each box in the figure is a representation of the chess board. Each digit in the boxes represents a square which is available for placement of a queen; each dot represents an unavailable square. The digits also represent the elements which remain in the variable ranges; e.g., if row 6 in a box contains 3, 7, and 8, then the range of \( S(6) \) contains 3, 7, and 8. The search begins with the box at the left of the top line in the figure and proceeds left-to-right across each line until a contradiction is encountered. Following a contradiction the search continues at the leftmost box of the next line down. The order in which variables were assigned values is \( S(8), S(7), \ldots, S(1) \).
Figure VII.19. Beginning of ARF's search for the 8 queens problem
Each time a value is assigned to some variable $s(j)$ during the search the remaining squares of row $j$ become automatically unavailable (i.e., $s(j)$ can have only one value). After each value assignment the phase 2 \textsc{vexclv} processor makes unavailable those squares in the same column and the same diagonals as the assigned square. For example, in the first box of Figure VII.19, $s(8)$ is assigned the value 1. As a result of this assignment the squares of row 8, column 1, and the lower left to upper right main diagonal are made unavailable. In the second box of the figure $s(7)$ is assigned the value 3. This assignment causes more squares to be marked unavailable. In the third box $s(6)$ is assigned the value 5. This initiates a deductive chain which leads to a contradiction as follows. The only available column in row 3 after the assignment of 5 to $s(6)$ is 4, so that 4 may be assigned as the value of $s(3)$. This assignment implies that 7 is the only available value for $s(1)$. This implies that only 2 is available for $s(2)$ and that only 8 is available for $s(5)$. These assignments leave no available value for $s(4)$, thereby producing the contradiction.

Figure VII.20 describes the entire backtrack search conducted by \textsc{arf} to find a solution. The digits in the figure indicate the value assignment made at each step in the search. The search
PROCEEDS TOP TO BOTTOM AND LEFT TO RIGHT AS IN FIGURE VII.19. ARF REQUIRED 10.7 MINUTES TO CONDUCT THE 44 STEP SEARCH. THE SOLUTION VALUES ARE AS FOLLOWS: S(8)=1, S(7)=5, S(6)=8, S(5)=6, S(4)=3, S(3)=7, S(2)=2, AND S(1)=4.
Figure VII.28. A tied backtracking search for the 8 queens problem
B. PROCESS CONSTRAINT SATISFACTION PROBLEMS

1. A CRYPT-ADDITION PROBLEM

Consider once more the crypt-addition problem introduced in Chapter II. The problem is one of assigning a decimal digit to each of the letters D, N, E, S, R, O, M, and Y such that no digit is assigned to more than one letter, neither S nor M is assigned zero, and the equation $\text{SEND} + \text{MORE} = \text{MONEY}$ is true when the letters are replaced by the digits assigned to them. In Chapter II we presented a ref statement of this problem which uses the column-by-column addition process to verify that the digits selected for each letter satisfy the equation $\text{SEND} + \text{MORE} = \text{MONEY}$. That ref procedure is shown in Figure II.6B.

ARP found a solution to this problem in the following manner. The `If` statement at line 13 of the ref procedure causes case analysis to occur because the value of its boolean expression depends on the values assigned to `select` function calls. Each time the `If` statement is interpreted, two context structures are output; since the `For` loop containing the `If` statement is interpreted four times, there are 16 possible contexts which can be interpreted to `End`. ARP's executive guides the interpretation of these contexts and produces the search tree shown in Figure VII.21. The nodes of this tree occur preceding each interpretation
OF THE VIFV STATEMENT. THE FIGURE SHOWS THE ORDER IN WHICH THE TREE IS GROWN, THE ELAPSED PROCESSING TIME AT EACH NODE, AND THE LAST STATEMENT INTERPRETED ON EACH TERMINAL BRANCH.

THE ONLY CONTEXT NOT ELIMINATED BY THE CONSTRAINT MANIPULATION METHODS BEFORE BEING INTERPRETED TO VENDV IS THE ONE CONTAINING THE SOLUTION. FIGURE VI.22 SHOWS THAT CONTEXT AT VENDV BEFORE THE BACKTRACKING SEARCH BEGINS. THE SEARCH ROUTINE ARBITRARILY SELECTS S(2) TO BE ASSIGNED A VALUE AND TRIES 2, 3, 4, AND 5 BEFORE SUCCEEDING WITH A VALUE OF 6. THIS ASSIGNMENT FORCES A VALUE OF 7 FOR S(1) AND PRODUCES THE SOLUTION SHOWN IN FIGURE VII.23.
Branching condition \(<2[1(0)]> + <2[1(0)]> + <2[1(0)]> < 2[0(1)]>\)
is assumed true on each T branch and false on each F branch.

Figure VII.21. Search tree for the crypt-addition problem
DATA STRUCTURE

\begin{align*}
\text{CARRY} & \quad <\text{CARRY}> : 1 \\
\text{Y} & \quad <\text{Y}> : -11 + S(2) + S(1) \\
\text{M} & \quad <\text{M}> : 1 \\
\text{O} & \quad <\text{O}> : 0 \\
\text{R} & \quad <\text{R}> : 9 \\
\text{S} & \quad <\text{S}> : 0 \\
\text{N} & \quad <\text{N}> : S(2) \\
\text{DUMMY} & \quad <\text{T}> : S(1) \\
\text{T} & \quad <\text{T}> : ? \\
\text{I} & \quad <\text{I}> : ? \\
\text{VECTOR: D, N, E, S, R, O, M, Y} \\
\text{SUM} & \quad \text{VECTOR: M, O, N, E, Y} \\
\text{A2} & \quad \text{VECTOR: X, M, O, R, E} \\
\text{A1} & \quad \text{VECTOR: X, S, E, N, D}
\end{align*}

VARIABLES

\begin{align*}
S(9) & \quad \text{VALUE: } -11 + S(2) + S(1) \\
\text{RANGE: 0 1 2 3 4 5 6 7 8 9} \\
S(7) & \quad \text{VALUE: 1} \\
S(6) & \quad \text{VALUE: 0} \\
S(5) & \quad \text{VALUE: 8} \\
S(4) & \quad \text{VALUE: 9} \\
S(3) & \quad \text{VALUE: } -1 + S(2) \\
\text{RANGE: 0 1 2 3 4 5 6 7 8 9} \\
S(2) & \quad \text{RANGE: 2 3 4 5 6 7 8 9} \\
S(1) & \quad \text{RANGE: 0 2 3 4 5 6 7 8 9}
\end{align*}

CONSTRAINTS

\begin{align*}
\text{EXCL}(1), S(2), 8 - S(4) - S(2), S(4), 17 - S(4), -9 + S(4), 1, -2 + S(4) + \\
S(2) + S(1)) \\
S(2) + S(1) & \leq 21 \\
10 & < S(2) + S(1)
\end{align*}

Figure VII.22. Context at end of interpretation
DATA STRUCTURE

J
  <J>: 5
CARRY
  <CARRY>: 1
Y
  <Y>: 2
M
  <M>: 1
O
  <O>: 0
R
  <R>: 8
S
  <S>: 9
F
  <F>: 5
N
  <N>: 6
DUMMY
D
  <D>: 7
I
  <I>: 2
L
  VECTOR: D,N,E,S,R,O,W,Y
SUM
  VECTOR: M,O,N,E,Y
A2
  VECTOR: X,M,O,R,E
A1
  VECTOR: X,S,E,N,D

VARIABLES
S(9)
  VALUE: 2
S(7)
  VALUE: 1
S(6)
  VALUE: 0
S(5)
  VALUE: 8
S(4)
  VALUE: 9
S(3)
  VALUE: 5
S(2)
  VALUE: 6
S(1)
  VALUE: 7

CONSTRAINTS: NONE

Figure VII.23. Solution context
THE CONTRADICTIONS FOUND BY THE CONSTRAINT MANIPULATION METHODS ARE AS FOLLOWS. AT CIRCLE 4 IN THE SEARCH TREE THE STATEMENT L3 PRODUCES THE CONSTRAINT \( vS(3) + S(6) = S(2) \) IN A CONTEXT WHICH HAS THE VALUE \( vS(3) - S(5) \) FOR \( S(2) \). THIS VALUE CAUSES THE CONSTRAINT TO REDUCE TO \( vS(5) + S(6) = 0 \). PROCESSING OF THIS CONSTRAINT CAUSES BOTH \( S(5) \) AND \( S(6) \) TO BE ASSIGNED ZERO AS A VALUE. THESE ASSIGNMENTS REVEAL THE INCONSISTENCY BY CAUSING THE \( \text{VEXCLV} \) CONSTRAINT FROM LINE 8 TO BECOME FALSE.

THE CONTRADICTIONS AT CIRCLES 6, 10, 17, AND 21 OCCUR BECAUSE THE \( \text{VCONDITIONV} \) STATEMENT AT LINE 21 OF THE REF PROCEDURE PRODUCES THE CONSTRAINT \( vS(7) = 0 \), AND 0 HAS BEEN ELIMINATED FROM THE RANGE OF \( S(7) \) IN EACH CONTEXT BY INTERPRETATION OF THE \( \text{VCONDITIONV} \) STATEMENT AT LINE 10.

AT CIRCLE 7 THE \( \text{VCONDITIONV} \) STATEMENT AT LINE 21 PRODUCES THE CONSTRAINT \( vS(7) = 1 \) IN A CONTEXT WHICH CONTAINS THE CONSTRAINTS \( v9 < S(4) + S(7) \) AND \( v18 < S(4) + S(7) + S(3) \). PROCESSING OF THE CONSTRAINT \( vS(7) = 1 \) IMPLIES A VALUE OF 1 FOR \( S(7) \) AND REDUCTION OF THE OTHER TWO CONSTRAINTS TO \( v8 < S(4) \) AND \( v17 < S(4) + S(3) \). CONSTRAINT \( v8 < S(4) \) IMPLIES A VALUE OF 9 FOR \( S(4) \) AND REDUCTION OF THE OTHER CONSTRAINT TO \( v8 < S(3) \). THIS CONSTRAINT IMPLIES A VALUE OF 9 FOR \( S(3) \) AND REVEALS AN INCONSISTENCY BY CAUSING THE \( \text{VEXCLV} \) CONSTRAINT FROM LINE 8 TO BECOME FALSE.
At Circle 11 the condition statement at line 21 produces the constraint \( vS(7) = 1 \) in a context which contains the constraint \( v3 < S(4) + S(7) \) and the value \( v19 - S(4) - S(7) \) for the variable \( S(5) \). Constraints \( vS(7) = 1 \) and \( v9 < S(4) + S(7) \) imply a value of 1 for \( S(7) \) and a value of 9 for \( S(4) \) as described in the previous paragraph. These values cause the value of \( S(5) \) to reduce to 9. This reveals an inconsistency as before by causing the \( \text{EXCLv} \) condition from line 8 to become false.

At Circle 12 the condition statement at line 15 produces the constraint \( vS(3) + S(6) = 10 + S(2) \) in a context which has a value of \( vS(2) + S(5) - 10 \) for \( S(3) \). The \( S(3) \) value causes the new constraint to reduce to \( vS(5) + S(6) = 20 \). Processing of this constraint causes the value \( v20 - S(6) \) to be assigned to \( S(5) \) and the creation of the constraints \( v-1 < 20 - S(6) \) and \( v20 - S(6) < 10 \). The constraint \( v20 - S(6) < 10 \) reveals an inconsistency by being false for all values in the range of \( S(6) \).

At Circle 15 the statement L3 produces the constraint \( vS(3) + S(6) = S(2) \) in a context which has a value of \( vS(3) - S(5) - 1 \) for \( S(2) \). This value for \( S(2) \) causes the new constraint to reduce to \( vS(5) + S(6) = -1 \). Processing of this constraint causes the value \( v-1 - S(6) \) to be assigned to \( S(5) \) and the creation of the constraints \( v-1 < -1 - S(6) \) and \( v-1 - S(6) < 10 \). The constraint
\[ v-1 < -1 - s(6) \] reveals an inconsistency by being false for all values in the range of \( s(6) \).

At circle 18 the \( v \) condition statement at line 21 produces the constraint \( v s(7) = 1v \) in a context which has the value \( v s(4) + s(7) - 9v \) for \( s(6) \) and the constraints \( v 8 < s(4) + s(7) v \) and \( v 18 < s(4) + s(7) + s(3) v \). The constraint \( v s(7) = 1v \) implies reduction of the \( s(6) \) value to \( v s(4) - 8v \) and the constraints to \( v 7 < s(4) v \) and \( v 17 < s(4) + s(3) v \). The constraint \( v 7 < s(4) v \) implies the value of \( s(4) \) must be either 8 or 9. The value 1 for \( s(7) \) and the value \( v s(4) - 8v \) for \( s(6) \) in the \( v \) exclamation constraint from line 8 imply that the value of \( s(4) \) cannot be 9. Assignment of 8 as the value of \( s(4) \) causes the other constraint to reduce to \( v 9 < s(3) v \). This constraint reveals an inconsistency by being false for all values in the range of \( s(3) \).

ARF required 7.9 minutes to find the solution. We gave this problem to the modified version of ARF described above in the discussion of the picnic problem. This version of ARF does not process the constraints in a context until that context has been interpreted to \( v \) end, and then uses the selection routine described in Chapter IV to determine the order in which the constraints are processed. This version of ARF found the solution in 6.8 minutes, a reduction of 14 per cent in processor time. The
USE OF THE SELECTION ROUTINE PRODUCES A LARGE REDUCTION IN THE TIME REQUIRED TO ELIMINATE EACH CONTEXT ONCE THE CONTEXT HAS BEEN INTERPRETED TO \texttt{END}, BUT MORE CONTEXTS MUST BE ELIMINATED SINCE NO ELIMINATION OCCURS PRIOR TO INTERPRETATION TO \texttt{END} AS BEFORE. THIS TRADEOFF PREVENTS A REDUCTION OF MORE THAN 14 PER CENT IN TOTAL PROCESSING TIME.

2. A HYPOTHESIS FORMATION PROBLEM


THE EXEMPLARS USUALLY HAVE AN EASILY RECOGNIZABLE SET OF ATTRIBUTES WHICH THE SUBJECT MAY USE IN FORMING A RULE. FOR
EXAMPLE, EACH EXEMPLAR MIGHT BE A CARD WITH A CIRCLE DRAWN ON IT. THE ATTRIBUTES RELEVANT FOR FORMING THE SETS COULD BE SIZE OF CIRCLE (LARGE OR SMALL), COLOR OF CIRCLE (RED OR BLUE), AND LOCATION OF CIRCLE (ON LEFT OR RIGHT SIDE OF THE CARD). THESE THREE ATTRIBUTES EACH HAVING TWO POSSIBLE VALUES ALLOW THE FORMATION OF 8 EXEMPLARS AND 255 HYPOTHESES. THE PROBLEM WE ARE CONSIDERING HAS THE SAME COMBINATORIAL CHARACTERISTICS AS THIS EXAMPLE. IN OUR PROBLEM EACH EXEMPLAR IS A QUADRUPLE WHOSE FIRST THREE ELEMENTS ARE EITHER 0 OR 1 AND WHOSE FOURTH ELEMENT IS EITHER Y OR N. THE FIRST THREE ELEMENTS REPRESENT THE BINARY VALUES OF THREE ATTRIBUTES AND THE FOURTH ELEMENT DENOTES WHICH SUBSET THE EXEMPLAR IS IN.

IN OUR STATEMENT OF THE PROBLEM IN FIGURE V.2A WE SPECIFY THE MANNER IN WHICH THE RULE OR HYPOTHESIS IS TO BE STATED. THE RULE IS TO DEFINE THE \( \text{vyv} \) SET AS A DISJUNCTION OF SETS WHICH HAVE CERTAIN COMMON ATTRIBUTE VALUES. ALL EXEMPLARS NOT IN THE \( \text{vyv} \) SET ARE ASSUMED TO BE IN THE \( \text{vNv} \) SET. WE ALLOW A MAXIMUM OF FIVE DISJUNCTS IN THE DEFINITION OF SET \( \text{vyv} \).

THE REF STATEMENT OF THIS PROBLEM SHOWN IN FIGURE V.2B DEFINES THE EXEMPLARS IN LINES 4 THROUGH 11, SELECTS A RULE IN LINES 12 THROUGH 14, AND THEN DETERMINES IF THE RULE CORRECTLY DIVIDES THE EXEMPLARS INTO THE SETS \( \text{vyv} \) AND \( \text{vNv} \).
ARP found a solution to this problem by conducting a lengthy search in the subproblem space. As noted in Chapter V, ARP's search is depth-first for this problem with the easiest contexts to eliminate being considered first at each node in the search tree. Figure VII.24 shows how the search progresses to the solution. Each box in the figures represents a context at a node in the search tree. The boxes contain the values remaining in the ranges of the variables with the first line of numbers being the range of variable one, the second line the range of variable two, etc. Variable one represents the value selected for \( <m> \) at line 12 of the REF procedure, variables two, three, and four define the set H1, and variables five, six, and seven define the set H2. The search begins with the box at the left of the top line in the figure and proceeds left-to-right across each line until a contradiction or \( \text{END} \) is encountered. Following the elimination of a context by a contradiction the search continues at the leftmost box of the next line down. Included at each point where a contradiction occurred is the constraint which caused the contradiction by reducing to false. The solution required 13.5 minutes of processing time.
Figure VII.24. IRT's search for the hypothesis formation problem.
ARF'S SEARCH STRATEGY OF ATTEMPTING TO ELIMINATE CONTEXTS AS SOON AS POSSIBLE IS EFFECTIVE THROUGHOUT THE SEARCH. THERE ARE 23 BRANCHING NODES IN THE SEARCH TREE AT WHICH TWO CONTEXTS ARE PRODUCED. FOR 12 OF THESE 23 NODES THE CONTEXT CHOSEN BY ARF IS ELIMINATED BEFORE THE NEXT BRANCHING NODE IS REACHED. THIS ELIMINATION OF CONTEXTS KEEPS THE AVERAGE NUMBER OF TERMINAL NODES IN THE TREE DURING THE SEARCH TO LESS THAN FIVE.

3. THE THREE COINS PUZZLE

WE NOW CONSIDER A TASK (FILIPiAK, 1942) WHICH BEGINS WITH THREE COINS SITTING ON A TABLE. BOTH THE FIRST AND THIRD COINS SHOW TAILS, WHILE THE SECOND COIN SHOWS HEADS. THE PROBLEM IS TO MAKE ALL THREE COINS THE SAME—EITHER HEADS OR TAILS—IN PRECISELY THREE MOVES. EACH MOVE CONSISTS OF TURNING OVER ANY TWO OF THE THREE COINS. FOR EXAMPLE, IF THE FIRST MOVE CONSISTED OF TURNING OVER THE FIRST AND THIRD COINS, THEN ALL OF THE COINS WOULD BE HEADS IN THE RESULTING SITUATION; BUT THE TASK IS NOT SOLVED BECAUSE ONLY ONE MOVE WAS TAKEN INSTEAD OF THE REQUIRED THREE.

WE PRESENT IN FIGURES VII.25A, VII.25B, AND VII.25C THREE FORMULATIONS OF THIS PROBLEM IN REF. IN THE FIRST FORMULATION, SHOWN IN FIGURE VII.25A, EACH COIN IS REPRESENTED BY AN INTEGER WHICH ACTS AS A POINTER TO AN ELEMENT OF A PARITY VECTOR. EACH
Element of the \textit{parity} vector is either \textit{heads} or \textit{tails} so that the element indicated by a coin in a container names the side of the coin which is showing. Each move is made by selecting the two coins that are to be moved. After the selection a \textit{condition} statement is used to assure that the selected coins are distinct. The move is then completed by adding 1 to the pointers which represent the selected coins. After three moves are made two \textit{condition} statements are encountered which require that the three coins all point to either \textit{heads} or \textit{tails}. 
BEGIN;
    SET_VECTOR PARITY TO HEADS, TAILS, HEADS, TAILS;
    SET_VECTOR COINS TO 1, 2, 1;
    FOR MOVES = 3 DO TO L1;
    SET <C1> TO SELECT(1, 3);
    SET <C2> TO SELECT(1, 3);
    CONDITION ~(<C1> = <C2>);
    SET COINS[<C1>] TO COINS[<C1>] + 1;
L1: SET COINS[<C2>] TO COINS[<C2>] + 1;
    CONDITION PARITY[COINS[1]] = PARITY[COINS[2]];
    CONDITION PARITY[COINS[2]] = PARITY[COINS[3]];
END;

FIGURE VII.25A. FIRST REP FORMULATION OF THE THREE COINS PUZZLE
BEGIN;

SET VECTOR COINS TO 1,0,1;
FOR MOVES = 3 DC. TO L1;
SET <COINS.MOVED> TO 0;
FOR I = 3 DO TO L2;
SET <TEMP> TO SELECT(0,1);
IF <TEMP> = COINS[I] THEN L2;
SET COINS[I] TO <TEMP>;
SET <COINS.MOVED> TO <COINS.MOVED> + 1;
L2:
L1: CONDITION <COINS.MOVED> = 2;
CONDITION COINS[1] = COINS[2];
CONDITION COINS[2] = COINS[3];
END;

FIGURE VII. 25B. SECOND REF FORMULATION OF THE THREE COINS PUZZLE
BEGIN;

SET.VECTOR PARITY TO HEADS, TAILS, HEADS, TAILS, HEADS;

SET.VECTOR COINS TO 1, 2, 1;

FOR MOVES - 3 DO TO L1;

SET.VECTOR MOVE TO SELECT (0, 1), SELECT (0, 1), SELECT (0, 1);


FOR I = 3 DO TO L1;

L1: SET COINS [I] TO COINS [I] + MOVE [I];

CONDITION PARITY [COINS [1]] = PARITY [COINS [2]];

CONDITION PARITY [COINS [2]] = PARITY [COINS [3]];

END;

FIGURE VII. 25C. THIRD REF FORMULATION OF THE THREE COINS PUZZLE
IN THE SECOND FORMULATION, SHOWN IN FIGURE VII.25B, EACH COIN IS REPRESENTED BY A BINARY DIGIT. THE DIGIT 1 DENOTES TAILS AND 0 DENOTES HEADS. THE MOVES ARE MADE IN TWO NESTED \texttt{FOR \_LOOPS}. TO MAKE A MOVE A NEW BINARY DIGIT IS SELECTED FOR EACH COIN. BEFORE THE NEW DIGIT IS ASSIGNED TO THE COIN A TEST IS MADE TO DETERMINE IF IT IS THE SAME AS THE EXISTING DIGIT. IF NOT, THEN THE COIN IS BEING TURNED OVER AND THE _\texttt{COINS \_MOVED}_ COUNTER IS STEPPED. AT THE COMPLETION OF A MOVE THIS COUNTER MUST INDICATE THAT TWO COINS HAVE BEEN TURNED OVER. AFTER THREE MOVES ARE MADE, TWO _\texttt{CONDITION}_ STATEMENTS ARE ENCOUNTERED WHICH REQUIRE THAT THE THREE COINS ALL BE EQUAL.

IN THE THIRD FORMULATION, SHOWN IN FIGURE VII.25C, THE REPRESENTATION OF THE COINS IS THE SAME AS IN THE FIRST FORMULATION. A MOVE IS MADE BY SELECTING 0 OR 1 FOR EACH COIN AND ADDING THE SELECTED DIGITS TO THE COIN POINTERS. HENCE, A COIN IS TURNED OVER DURING A MOVE ONLY IF 1 IS THE DIGIT SELECTED FOR IT. A _\texttt{CONDITION}_ STATEMENT IS USED TO ASSURE THAT TWO COINS ARE TURNED OVER DURING EACH MOVE BY REQUIRING THAT THE SUM OF THE SELECTED DIGITS BE 2. AFTER THREE MOVES THE FINAL CONDITIONS ARE STATED AS IN THE FIRST FORMULATION.

WE PRESENT THESE FORMULATIONS TO ILLUSTRATE THAT THE DIFFICULTY OF FINDING A SOLUTION TO A PROBLEM CAN BE DEPENDENT NOT
ONLY ON THE PROBLEM BUT ALSO ON HOW THE PROBLEM IS STATED. USING EITHER OF THE FIRST TWO FORMULATIONS ARF WOULD NOT BE ABLE TO FIND A SOLUTION IN A REASONABLE LENGTH OF TIME, BUT WHEN ARF WAS GIVEN THE THIRD FORMULATION IT FOUND A SOLUTION IN 2.1 MINUTES OF PROCESSING TIME.

THE REASON FOR THIS DIFFERENCE IN DIFFICULTY IS THAT ARF MUST DO A LARGE AMOUNT OF CASE ANALYSIS WHEN GIVEN EITHER OF THE FIRST TWO FORMULATIONS, WHEREAS NO CASE ANALYSIS AT ALL IS REQUIRED WITH THE THIRD FORMULATION. GIVEN THE FIRST FORMULATION ARF MUST GENERATE NEW CASES EACH TIME IT INTERPRETS THE \texttt{SETV} STATEMENTS AT LINES 8 AND 9 OF THE PROCEDURE. THESE TWO STATEMENTS WILL CAUSE INTERPRETATION OF THE L1 LOOP TO PRODUCE 216 CASES. IN THE SECOND FORMULATION ARF MUST GENERATE A NEW CASE EACH TIME THE \texttt{IFV} STATEMENT IS INTERPRETED. THIS STATEMENT WILL CAUSE INTERPRETATION OF THE L1 LOOP TO PRODUCE 27 CASES. FOR BOTH THESE FORMULATIONS MOST OF THE CASES WOULD HAVE TO BE GENERATED AND CONSIDERED SINCE WHEN NEW VARIABLES ARE BEING CONTINUALLY DEFINED DURING INTERPRETATION ARF SEARCHES IN A BREADTH-FIRST MANNER.

WE OBSERVED IN CHAPTER III THAT CASE ANALYSIS IS REQUIRED WHEN THE EVALUATION OF A SLOT EXPRESSION OR OF THE EXPRESSION IN AN \texttt{IFV} OR COMPUTED \texttt{GOTOV} STATEMENT DEPENDS ON THE VALUE OF A VARIABLE. IN ACCORDANCE WITH THIS OBSERVATION THE CASE ANALYSIS IN
THE FIRST FORMULATION OF THE THREE COINS TASK IS CAUSED BY THE SLOT EXPRESSIONS \( \text{\texttt{\textless C1\textgreater }} \text{\texttt{\textbackslash coin}} \) AND \( \text{\texttt{\textless C2\textgreater }} \text{\texttt{\textbackslash coin}} \), AND IN THE SECOND FORMULATION IS CAUSED BY THE \( \text{\texttt{\textless I\textgreater \texttt{\textbackslash if}}} \) STATEMENT'S BRANCHING CONDITION \( \text{\texttt{\textless I\textgreater \texttt{\textbackslash temp}}} \geq \text{\texttt{\textless I\textgreater \texttt{\textbackslash coins}}} \).

ARFVS SOLUTION PROCESS USING THE THIRD FORMULATION IS STRAIGHTFORWARD. FIGURE VII.26 SHOWS THE CONTEXT AFTER THE THREE MOVES HAVE BEEN MADE. THE \( \text{\texttt{\textless I\textgreater \texttt{\textbackslash condition}}} \) STATEMENT AT LINE 6 OF THE PROCEDURE IS USED TO ELIMINATE THREE OF THE VARIABLES AND CREATE SIX NEW CONSTRAINTS. WHEN THE CONTEXT IS MOVED TO \( \text{\texttt{\textless I\textgreater \textbackslash vend}} \) THE BACKTRACKING SEARCH FINDS A SOLUTION AFTER SUCCESSFULLY ASSIGNING 0 AS THE VALUE OF \( \text{\texttt{\textless I\textgreater \texttt{\textbackslash s(3)}}}, \text{\texttt{\textbackslash s(6)}} \), AND \( \text{\texttt{\textbackslash s(8)}} \). THE CONTEXT CONTAINING THE SOLUTION IS SHOWN IN FIGURE VII.27.
CONTEXT 16114

DATA STRUCTURE
  
MOVE
  <MOVE>: 4
  
MOVES
  <MOVES>: 4

COINS
  <COINS>: 4
  
PARITY
  <PARITY>: 4

VARIABLES
  S(9)
    VALUE: 2+-S(8)+-S(9)
    RANGE: 0 1

  S(8)
    RANGE: 0 1

  S(7)
    VALUE: 2+-S(8)+-S(9)
    RANGE: 0 1

  S(6)
    RANGE: 0 1

  S(5)
    RANGE: 0 1

  S(4)
    VALUE: 2+-S(5)+-S(6)
    RANGE: 0 1

  S(3)
    RANGE: 0 1

  S(2)
    RANGE: 0 1

  S(1)
    VALUE: 2+-S(2)+-S(3)
    RANGE: 0 1

CONSTRAINTS
  -S(8)+-S(9)<0
  -3<-S(8)+-S(9)
  -S(5)+-S(6)<0
  -3<-S(5)+-S(6)
  -S(7)+-S(3)<0
  -3<-S(2)+-S(3)

Figure VII.26. Context after the three moves
Figure VII.27. Solution context
C. HEURISTIC SEARCH PROBLEMS

1. THE MONKEY PROBLEM

Consider again the monkey problem introduced in Chapter II. The problem originates in the study of the problem solving ability of primates and involves a monkey's attempt to reach some bananas hanging from the ceiling of his cage. To get the bananas he must move a box from another part of the cage to under the bananas, climb on the box, and pull them down.

We presented in Figure II.98 a statement of the monkey problem in Ref. Note that this Ref Procedure contains no \texttt{\textbf{condition}} statements. The Procedure's control structure is designed so that only those options which are actually available to the monkey at any given time may be selected. For example, at the beginning of the problem the monkey's only active option is to walk. This is reflected in the Ref Procedure in that interpretation proceeds automatically to the \texttt{\textbf{walk}} statement and the monkey's first selection involves only where he is to walk. This absence of \texttt{\textbf{condition}} statements implies that the constraint manipulation methods will not be able to eliminate any contexts during interpretation so that at each step every terminal node of the search tree is extendable and must be considered by the object
SELECTION ROUTINE.


AS DISCUSSED IN CHAPTER V, ARF CONDUCTS A BASICALLY BREADTHFIRST SEARCH WHEN GIVEN A PROBLEM SUCH AS THIS ONE WHERE NEW VARIABLES ARE CONTINUALLY BEING DEFINED DURING INTERPRETATION. SINCE THE SOLUTION TO THIS PROBLEM REQUIRES FIVE VARIABLES, ARF'S SEARCH EXECUTIVE WILL NOT CONSIDER INTERPRETING THE SOLUTION CONTEXT TO VENDV UNTIL ALL CONTEXTS IN THE SEARCH TREE HAVE AT LEAST FOUR VARIABLES. THE ENTIRE SOLUTION PROCESS REQUIRED 3.4 MINUTES.
2. A WATERJUG PROBLEM

Consider again the waterjug problem introduced in Chapter II. The goal of this problem is to put exactly two gallons of water into a five gallon jug given only the five gallon jug, an eight gallon jug, a water source, and a water sink. In Chapter II we discussed this problem as a heuristic search problem and presented a ref statement of it in Figure II.88. Lines 6 through 23 of the ref procedure define the six actions available to the problem solver, and interpretation of the procedure proceeds by repeatedly selecting an action, carrying out the selected action, and testing for the desired final state.

ARF would not be able to solve this problem in a reasonable length of time because of the large amount of case analysis necessary during the interpretation. Interpretation of the computed \texttt{GOTO} statement at line 5 of the procedure produces six cases and since this statement must be interpreted four times before a solution is possible, the search tree could have as many as 614 or 1296 terminal nodes. Because there are no \texttt{CONDITION} statements in this procedure, none of the cases can be eliminated during the search. In addition, the constraints produced during the interpretation of the computed \texttt{GOTO} statement reduce the range of each variable to one element so that ARF's use of
SOLUTION SPACE SIZE FOR GUIDING THE SEARCH IS ENTIRELY INEFFECTUAL.

ARF COULD SOLVE THIS PROBLEM WITH A SMALL AMOUNT OF EFFORT IF IT HAD THE CAPABILITY FOR RECOGNIZING LOOPS DESCRIBED IN CHAPTER V. WITH THIS CAPABILITY THERE WOULD BE TWO CASES AT THE FIRST INTERPRETATION OF STATEMENT L7, THREE CASES AT THE SECOND INTERPRETATION, TWO CASES AT THE THIRD INTERPRETATION, AND TWO CASES AT THE FOURTH INTERPRETATION, ONE OF WHICH WOULD CONTAIN THE SOLUTION.

3. THE TOWER OF Hanoi PROBLEM

WE NOW CONSIDER A CLASSICAL PUZZLE WHICH CONSISTS OF THREE PEGS AND A NUMBER OF DISKS, EACH OF WHOSE DIAMETER IS DIFFERENT FROM ALL OF THE OTHERS. INITIALLY, ALL OF THE DISKS ARE STACKED ON THE FIRST PEG IN ORDER OF DESCENDING SIZE. THE PROBLEM IS TO DISCOVER A SEQUENCE OF MOVES WHICH WILL TRANSFER ALL OF THE DISKS TO THE THIRD PEG. EACH MOVE CONSISTS OF REMOVING THE TOP DISK ON ANY PEG AND PLACING IT ON TOP OF THE DISKS ON ANOTHER PEG, BUT NEVER PLACING A DISK ON TOP OF ONE SMALLER THAN ITSELF.

FIGURE VII.29 SHOWS A REP STATEMENT OF THE FOUR DISK VERSION OF THIS PROBLEM. IN THIS STATEMENT THE PEGS ARE REPRESENTED BY THE VECTORS ASSOCIATED WITH PEG1, PEG2, AND PEG3. FOR EACH PEG, \langle PEG \rangle
INDICATES THE NUMBER OF DISKS ON PEGJ; IN ADDITION, WHEN <PEGJ> HAS A POSITIVE VALUE IT POINTS TO THE TOP DISK ON PEGJ. FOR EXAMPLE, THE PROBLEM BEGINS WITH <PEG1>=4 WHICH INDICATES THAT THE FIRST PEG HAS FOUR DISKS ON IT AND THAT THE TOP DISK IS AT PEG1[4]. THE DISKS ARE REPRESENTED BY THE INTEGERS 1, 2, 3, AND 4, WITH 1 BEING THE SMALLEST DISK AND 4 BEING THE LARGEST.
BEGIN;

SET VECTOR PEG TO PEG1, PEG2, PEG3;
SET VECTOR PEG1 TO 4, 3, 2, 1;
SET <PEG1> TO 4;
SET <PEG2> TO 0;
SET <PEG3> TO 0;

L1: SET <FROM. PEG> TO PEG(SELECT (1, 3)];
CONDITION ~(<<FROM. PEG>> = 0);
SET <TO. PEG> TO PEG(SELECT (1, 3)];
CONDITION ~(<<FROM. PEG> = <TO. PEG>);
CONDITION <<TO. PEG>> = 0 \vee <FROM. PEG>[<<FROM. PEG>>] < <TC. PEG>[<<TO. PEG>>];
SET <<TO. PEG>> TO <<TO. PEG>> + 1;
SET <TO. PEG>[<<TO. PEG>>] TO <FROM. PEG>[<<FROM. PEG>>];
SET <<FROM. PEG>> TO <<FROM. PEG>> - 1;
IF ~(<PEG3> = 4) THEN L1;

END;

FIGURE VII.29. REF STATEMENT OF THE TOWER OF HANOI PROBLEM
In lines 1-6 of the procedure the initial state is defined.

At line 7 a peg is selected from which a disk is to be taken. Line 8 requires that the selected peg not be empty. At line 9 a peg is selected on which a disk will be placed. Line 10 requires that the two selected pegs be distinct, and line 11 requires that the move be a legal one by the rules of the problem. The move is made in lines 13-15, and a test for completion is made at line 16.

ARF could not solve this problem given the statement of Figure VII.29 because of the large number of cases produced during interpretation. The case analysis is required during each interpretation of the \texttt{SET} statements at lines 13 and 15. Since the interpreter must explicitly identify \texttt{<TO.PEG>} at line 13 and \texttt{<FROM.PEG>} at line 15, separate cases must be created for each possible combination of selections.

To solve this problem ARF would need the loop recognition capability described in Chapter IV and some means of directing the search. Loop recognition would eliminate most of the search tree, but since the solution requires 15 disk transfers, the remaining tree would still have 16 terminal nodes and 80 total nodes.
4. THE MISSIONARIES AND CANNIBALS PROBLEM

Consider the missionaries and cannibals problem which was discussed in Chapter III. This is a puzzle of some difficulty to humans which fits naturally into the heuristic search paradigm. It is often considered in discussions concerning heuristic problem solving programs (see Ernst and Newell, 1967, and Amarel, 1968).

The REP statement of this problem given in Figure III.3B has a form similar to the general form for REP statements of heuristic search problems shown in Figure II.7. The initial state is defined in lines 1 through 7 of the procedure; an operator is selected in lines 8 and 11; lines 9, 12-15, and 24-27 contain the applicability tests; the selected operator is applied in lines 16-23 and 28-30; and the test for the desired final state is at line 31.

ARF relies entirely on its constraint satisfaction methods to attempt the solution of this problem. The only case analysis required is at the VIP statement which tests for the final state. If the constraint manipulation methods cannot deduce a contradiction when the constraint vMISSIONARIES OF RIGHT SIDE = 3 ^ CANNIBALS OF RIGHT SIDE = 3 is added to the context which does not branch to statement L1, then two cases must be considered. But
As soon as the case at \texttt{VENDV} is considered again it either becomes a solution or is eliminated.

ARF cannot solve this problem in a reasonable amount of processing time. The primary difficulty is the large amount of constraint manipulation required in the solution attempt. Each boat trip creates two new variables and as many as seven new constraints. Since the solution requires eleven boat trips ARF will try eleven times to solve a constraint satisfaction problem which contains the constraint defining the desired final state. After the eleventh trip the context will have twenty-two variables and as many as seventy-seven constraints. The constraint manipulation methods in the current ARF are too slow to satisfy these requirements for finding a solution.

Figure VII.30 shows the context produced by ARF at the completion of the fourth boat trip. 4.5 minutes of processing time were required to produce this context. Note that constraint manipulation methods have not been able to eliminate any variables or to reduce any of the variable ranges.

The estimating facilities discussed in Chapter III would help ARF recognize that more processing time is being spent on constraint manipulation than can possibly be saved during the backtrack search in the comparatively small search space. This
RECOGNITION WOULD CAUSE THE PROCESSING OF NEW CONSTRAINTS TO BE BYPASSED THEREBY GREATLY SPEEDING UP THE SOLUTION ATTEMPT.
DATA STRUCTURE

<TEMP>: RIGHT_ST0P

<ARRIVING_SIDE>: RIGHT_ST0P

<DEPARTING_SIDE>: LEFT_SIDE

RIGHT_SIDE

CANNIBALS OF RIGHT_SIDE: S(2)+S(4)+S(6)+S(9)
MISSIONARIES OF RIGHT_SIDE: S(1)+S(3)+S(5)+S(7)

LEFT_SIDE

CANNIBALS OF LEFT_SIDE: 3+S(2)+S(4)+S(6)+S(9)
MISSIONARIES OF LEFT_SIDE: 3+S(1)+S(3)+S(5)+S(7)

VARIABLES

S(9)
RANGE: 0 1 2

S(7)
RANGE: 0 1 2

S(6)
RANGE: 0 1 2

S(5)
RANGE: 0 1 2

S(4)
RANGE: 0 1 2

S(3)
RANGE: 0 1 2

S(2)
RANGE: 0 1 2

S(1)
RANGE: 0 1 2

CONSTRAINTS

S(1)+S(3)+S(5)+S(7) = \forall \ 1 < S(2) + S(4) + S(6) + S(9) + S(1) + S(3) +
S(5) + S(7)

S(1)+S(3)+S(5)+S(7) = \exists \ -1 < S(2) + S(4) + S(6) + S(9) + S(1) + S(3) +
S(5) + S(7)

Figure VII.30. Context after four boat trips
Figure VII.30 (continued)
VIII. OVERVIEW AND CONCLUSIONS

We began by introducing the idea that a nondeterministic programming language could be used as a problem statement language for a general problem solving program. Nondeterministic languages represent a compromise between the ease of problem statement offered by a natural language such as English and the ease of problem solution offered by an axiomatic language such as the predicate calculus. The syntax and semantics of a nondeterministic programming language are formally defined so that problem statements can be algorithmically interpreted, yet the full representational power of the base programming language's control and data structures are available for describing the objects, relations, and processes involved in the statement of a problem.

By considering a particular example of a nondeterministic language, (i.e., REP), we have demonstrated how boolean constraint satisfaction problems, process constraint satisfaction problems, and heuristic search problems can be stated in a natural manner. We also indicated that suitable facilities could be included in REP for representing other classes of problems such as optimization problems.

Programming languages such as ALGOL and FORTRAN continually evolve in the direction of providing more
representational power to the user, and the addition of nondeterministic features to these languages would be another step in that evolution. The addition of such features impose new requirements on the problem solving capabilities of the systems which interpret these languages. For example, just as we assume that a standard VALGOL system can solve the problem of evaluating an algebraic expression or the problem of fetching the value of an array element, we would want a nondeterministic VALGOL system to solve the problem of finding values which satisfy a set of constraints or the problem of finding an execution path through a procedure containing nondeterministic branches.

In chapter III we introduced ARF, a program for solving problems stated in REF. A fundamental goal in the design of ARF was to translate a problem stated as a REF procedure into a form that would allow the application of effective problem solving methods. This goal was achieved by using variables to represent the values of VSELECTV function calls during the interpretation of a procedure. This use of variables eliminates the necessity for considering a separate interpretation case for each possible value of a VSELECTV function call and allows the interpreter to derive a set of Boolean expressions which constrain the values of the variables. The translated problem consisting of a set of variables, a finite range of possible values for each variable,
AND A SET OF CONSTRAINTS ON THE VALUES OF THOSE VARIABLES IS IN A FORM TO WHICH ANY OF A VARIETY OF CONSTRAINT SATISFACTION PROBLEM SOLVING METHODS CAN BE APPLIED.

THE USE OF VARIABLES IS NOT ALWAYS EFFECTIVE IN ELIMINATING THE NEED FOR CONSIDERING MULTIPLE CASES DURING INTERPRETATION. WHEN THIS CASE ANALYSIS IS REQUIRED ARF NEEDS METHODS FOR GUIDING THE INTERPRETER AS TO WHICH CASE TO PURSUE FIRST, AND ONCE A CASE IS CHOSEN, HOW LONG TO CONTINUE ITS INTERPRETATION BEFORE RETURNING TO CONSIDER ANOTHER ONE. THE ORGANIZATION OF ARF ALLOWS THE INTERPRETATION OF EACH CASE TO PROCEED IN SMALL INDEPENDANT STEPS SO THAT AN EXECUTIVE ROUTINE CAN USE HEURISTIC SEARCH METHODS TO GUIDE THE INTERPRETATION WITH THE GOAL OF INTERPRETING A CASE (I.E., CONTEXT) TO THE VENVV STATEMENT.

HENCE, ARF COMBINES CONSTRAINT SATISFACTION METHODS AND HEURISTIC SEARCH METHODS TO SOLVE A PROBLEM. A SEARCH IS CONDUCTED TO FIND A CASE WHICH CAN BE INTERPRETED TO VENVV, AND THE SIZE OF THIS SEARCH SPACE IS REDUCED BY THE USE OF VARIABLES TO ALLOW THE REPRESENTATION OF MANY CASES AS A SINGLE CASE. FOR A CASE IN THIS REDUCED SEARCH SPACE TO REPRESENT A SOLUTION, CONSTRAINT SATISFACTION METHODS MUST BE APPLIED TO FIND ACCEPTABLE VALUES FOR ITS VARIABLES. THIS AMOUNTS TO DETERMINING WHICH OF THE MANY CASES REPRESENTED BY THE ONE CASE IS A SOLUTION.
IN CHAPTER IV WE DISCUSSED THE DESIGN OF METHODS FOR SOLVING THE CONSTRAINT SATISFACTION PROBLEMS CREATED BY ARF's INTERPRETER. THE MOST POWERFUL OF THESE METHODS PERFORM ALGEBRAIC MANIPULATIONS ON CONSTRAINTS TO DEDUCE THAT A VARIABLE CAN BE EXPRESSED AS A FUNCTION OF OTHER VARIABLES, THAT AN ELEMENT CAN BE DELETED FROM THE RANGE OF A VARIABLE, OR THAT NO SET OF VALUES EXIST WHICH CAN SATISFY THE SET OF CONSTRAINTS. THESE ARE ARF's MOST POWERFUL PROBLEM SOLVING METHODS AND ARF IS MOST EFFECTIVE WHEN THE INTERPRETER PRODUCES ONLY A SMALL NUMBER OF CASES SO THAT THESE METHODS CAN PLAY THE PRIMARY ROLE IN SOLVING A PROBLEM.

IN CHAPTER V WE CONSIDERED ALTERNATIVE FORMULATIONS OF THE SEARCH PROBLEM ENCOUNTERED DURING THE INTERPRETATION OF A REP PROCEDURE AND PROPOSED HEURISTIC METHODS FOR GUIDING THE SEARCH. WE SHOWED THAT THE CONSTRAINT SATISFACTION METHODS DESCRIBED IN CHAPTER IV COULD BE FORMULATED AS OPERATORS IN THE SEARCH SPACE SO THAT ALL THE PROGRAM's PROBLEM SOLVING ACTIVITIES WOULD BE CONTROLLED BY A SINGLE SEARCH EXECUTIVE ROUTINE. THIS EXECUTIVE COULD FREELY INTERMIX INCREMENTAL STEPS OF INTERPRETATION, CONSTRAINT PROCESSING, AND VALUE ASSIGNMENT TO VARIABLES IN ITS SEARCH FOR A SOLUTION.

THE MOST SIGNIFICANT WEAKNESSES IN ARF's SEARCH STRATEGY ARE ITS INABILITY TO RECOGNIZE WHEN TWO OBJECTS ARE SEMANTICALLY
EQUIVALENT AND ITS INABILITY TO DO MEANS-ENDS ANALYSIS. FIVE
PROBLEM STATEMENTS WERE PRESENTED IN CHAPTER VII WHICH ARF CANNOT
SOLVE. IF THE EQUIVALENCE TEST FOR OBJECTS (I.E., CONTEXTS)
PROPOSED IN CHAPTER V WERE IMPLEMENTED ARF WOULD BE ABLE TO SOLVE
FOUR OF THOSE FIVE PROBLEMS AND WOULD BE ABLE TO SOLVE THE MONKEY
PROBLEM WITH MUCH LESS EFFORT.

THE PRIMARY REASON ARF CANNOT DO MEANS-ENDS ANALYSIS IS THAT
IT LACKS INFORMATION ABOUT THE FINAL OBJECT (I.E., GOAL) TO WHICH
THE SEARCH IS DIRECTED. THE ONLY KNOWN ATTRIBUTES OF THE FINAL
OBJECT ARE THAT IT HAS BEEN INTERPRETED TO END\text{\textsuperscript{\textvisiblespace}} AND ITS
CONSTRAINTS ARE SATISFIED. TO REMEDY THIS PROBLEM WE PROPOSED A
BI-DIRECTIONAL SEARCH USING AN INVERSE INTERPRETER. THE GOAL OF
THE BI-DIRECTIONAL SEARCH IS TO CONNECT THE TWO SEARCH TREES BY
COMBINING A TERMINAL NODE FROM ONE TREE WITH A TERMINAL NODE FROM
THE OTHER TREE. WITH THIS AS A GOAL IT IS POSSIBLE TO PROVIDE A
MEASURE OF THE \text{\textsuperscript{\textvisiblespace}}DISTANCE\text{\textsuperscript{\textvisiblespace}} BETWEEN TERMINAL NODES IN THE TWO
TREES AND TO DETERMINE WHAT CHANGES NEED TO OCCUR FOR A SUCCESSFUL
COMBINATION TO BE POSSIBLE. THE SEARCH CAN THEN BE DIRECTED TOWARD
MAKING THE CHANGES REQUIRED TO REDUCE THE DISTANCE BETWEEN THE
CLOSEST TERMINAL NODES.

A REOCCURRING THEME IN OUR DISCUSSIONS OF ARF'S PROBLEM
SOLVING ABILITIES HAS BEEN THAT BY INCREASING THE PROGRAM'S
FLEXIBILITY AND FREEDOM OF CHOICE WE CAN INCREASE THE POWER AND EFFECTIVENESS OF ITS PROBLEM SOLVING METHODS. THE FIRST OCCURRENCE OF THIS IDEA WAS THE INTRODUCTION OF VARIABLES TO REPRESENT SELECTIONS. THE VARIABLES ELIMINATE THE NECESSITY OF HAVING TO DETERMINE A SELECTION VALUE AT THE TIME THE SELECT FUNCTION IS CALLED AND PROVIDE THE FREEDOM TO DECIDE WHEN AND HOW THE VALUE WILL BE FOUND.

IN THE DISCUSSION OF CONSTRAINT SATISFACTION METHODS THE ISSUE OF FREEDOM OCCURRED AGAIN WHEN WE PROPOSED ALLOWING THE PROGRAM TO DETERMINE THE ORDER IN WHICH CONSTRAINTS ARE PROCESSED AND THE ORDER IN WHICH VARIABLES ARE ASSIGNED VALUES DURING THE BACKTRACKING SEARCH. WE THEN PROPOSED THAT AN ESTIMATING SCHEME BE USED TO PROVIDE THE OPTION OF NOT APPLYING THE CONSTRAINT MANIPULATION METHODS TO A CONSTRAINT.

IN THE DISCUSSION OF ALTERNATIVE FORMULATIONS OF THE SEARCH PROBLEM DURING INTERPRETATION WE NOTED THAT A SEARCH EXECUTIVE ROUTINE COULD BE MORE EFFECTIVE IF IT HAD THE FREEDOM TO CONTROL THE PROGRESS OF THE SEARCH AFTER INTERPRETATION OF EACH RESTATEMENT. WE ALSO PRESENTED A FORMULATION OF THE SEARCH PROBLEM IN WHICH THE SEARCH EXECUTIVE WOULD HAVE FREEDOM TO INTERMIX CONSTRAINT PROCESSING, VARIABLE VALUE ASSIGNMENT, AND INTERPRETATION STEPS.
BY PROVIDING THESE DEGREES OF FREEDOM IN THE DESIGN OF A
PROBLEM SOLVING PROGRAM, METHODS CAN BE INCLUDED WHICH RESPOND TO
THE CHARACTERISTICS OF EACH PARTICULAR PROBLEM. THIS ALLOWS THE
PROGRAM TO EXHIBIT THE VARIETY OF BEHAVIOR NECESSARY TO BE
EFFECTIVE AT SOLVING DIVERSE CLASSES OF PROBLEMS.

THERE ARE SEVERAL POSSIBILITIES FOR EXTENDING THE WORK WITH
REF AND ARF. IF WE WISH SYSTEMS LIKE ARF TO BE USEFUL PROBLEM
SOLVING ASSISTANTS, THEN WE NEED TO PROVIDE AS MUCH FLEXIBILITY IN
THE INPUT LANGUAGE AS POSSIBLE. BESEDES THE OBVIOUS ADDITIONAL
FEATURES NEEDED IN REF SUCH AS MULTIPLICATION, EXPONENTIATION,
REAL VALUED VARIABLES, AND BLOCK STRUCTURE, AN EXTENSION
CAPABILITY WOULD BE IMPORTANT SO THAT THE USER COULD DEFINE HIS
OWN DATA STRUCTURES AND OPERATORS. THE USER SHOULD ALSO BE ALLOWED
TO TELL THE SYSTEM ABOUT THE PROPERTIES OF NEWLY DEFINED
OPERATORS. HE COULD DO THIS BY IMPUTING IDENTITIES WHICH THE
SYSTEM COULD USE IN ITS EXPRESSION REDUCTION AND CONSTRAINT
MANIPULATIONS.

THE INPUT LANGUAGE SHOULD ALSO ALLOW THE USER TO DEFINE
OBJECTS AND RELATIONSHIPS AMONG OBJECTS AS IS DONE WITH
QUESTION-ANSWERING PROGRAMS. REF \texttt{VSET\texttt{v}} STATEMENTS COULD BE USED TO
BUILD UP THIS STORE OF INFORMATION. THE SYSTEM COULD CONSTRUCT A
\texttt{VMASTER CONTEXT\texttt{v}} CONTAINING THIS INFORMATION WHICH WOULD BE
CONSIDERED PART OF EACH CONTEXT USED DURING INTERPRETATION OF A
REF PROCEDURE. (NOTE, THIS WOULD ALLEVIATE THE BACKWARD
INTERPRETER'S INITIALIZATION PROBLEM DISCUSSED IN CHAPTER V.)

THESE EXPANSIONS OF THE INPUT LANGUAGE FOSE NEW DIFFICULTIES
IN THE DESIGN OF THE PROBLEM SOLVER. THE PROBLEM OF SATISFYING
CONSTRAINT SETS CONTAINING REAL VALUED VARIABLES IS ONE WHICH WILL
REQUIRE ACCESS TO A LIBRARY OF NUMERICAL ANALYSIS METHODS. ALSO,
THE EXPRESSION SIMPLIFICATION AND CONSTRAINT MANIPULATING METHODS
MUST BE MADE MORE SOPHISTICATED WHEN REAL VALUED VARIABLES AND
ADDITIONAL ARITHMETIC OPERATORS ARE ALLOWED.

IF THE USER CREATES NEW OPERATORS AND INPUTS IDENTITIES TO
DEFINE THEIR PROPERTIES, THEN THE SYSTEM WILL NEED A THEOREM
PROVING CAPABILITY TO MAKE USE OF THESE IDENTITIES IN SIMPLIFYING
EXPRESSIONS AND FORMING DEDUCTIONS FROM CONSTRAINTS.

THESE AND OTHER POSSIBILITIES SEEM WORTH PURSUING WITH THE
GOALS OF LEARNING MORE ABOUT THE NATURE OF INTELLIGENT PROBLEM
SOLVING ACTIVITY AND OF BUILDING A TRULY USEFUL GENERAL PROBLEM
SOLVING SYSTEM.
APPENDIX I: ACTIONS FOR EXPRESSION SIMPLIFICATION

The following are the action routines used by ARF to evaluate and simplify expressions. The control flow of the simplification executive and the behavior of the actions are discussed in Chapter IV. In describing the actions we use a prefix notation for representing expressions in which the operator is followed by a parenthesized list of its operands separated by commas. Each action is assumed to exit + unless there is an indication otherwise.

ELEM --- Action list contains one action, R137.

R137. If y is the value of X[J] in the context, then ELEM(SYMB(X), INTJ)) reduces to Y. Exit is - when the action is applicable.

OF --- Action list contains one action, R136.

R136. If z is the value of attribute X of identifier Y in the context, then OF(SYMB(X), SYMB(Y)) reduces to Z. Exit is - when the action is applicable.

SELE --- Action list contains one action, R139.

R139. If x is the value of variable S(I) in the context, then SELE(INT(I)) reduces to X. Exit is - when the action is applicable.

MINUS --- Action list contains R18, R14, and R116.

R18. If x is any expression, then -(+(...,X,...)) reduces to +(...,-(X),...). Each newly created minus expression, i.e. the -(X), is reduced via a recursive call of the simplification executive from within the action. Exit is - when the action is applicable.
R14. IF I IS ANY INTEGER THEN -(INTE(I)) REDUCES TO INTE(-I). EXIT IS - WHEN THE ACTION IS APPLICABLE.

R116. IF X IS ANY EXPRESSION, THEN -(-X)) REDUCES TO X. EXIT IS - WHEN THE ACTION IS APPLICABLE.

PLUS --- ACTION LIST CONTAINS R35, R13, AND R15.

R35. IF X AND Y ARE ANY EXPRESSIONS, THEN +(....,+(X,....,Y),....) REDUCES TO +(....,X,....,Y,....).

R13. IF I IS AN INTEGER AND X IS NOT AN INTE EXPRESSION, THEN +(X,...,INTE(I),....) REDUCES TO +(INTE(I),X,...). IF I AND J ARE INTEgers SUCH THAT I+J=0, THEN +(INTE(I),....,INTE(J),....) REDUCES TO +(INTE(I+J),....). IF I AND J ARE INTEgers SUCH THAT I+J=0, THEN +(INTE(I),....,INTE(J),....) REDUCES TO +(....). REDUCTION CONTINUES IN THIS MANNER UNTIL THE ACTION UNTIL ALL INTEGER OPERANDS HAVE BEEN COMBINED. IF THE RESULTING EXPRESSION HAS BUT ONE OPERAND, THEN THAT OPERAND IS OUTPUT AND EXIT IS -. IF THE RESULTING EXPRESSION HAS NO OPERANDS, THEN INTE(O) IS OUTPUT AND EXIT IS -. OTHERWISE EXIT IS +.

R13. IF X IS ANY EXPRESSION, THEN +(....,X,....,-(X),....) REDUCES TO +(....). IF THE RESULTING EXPRESSION HAS BUT ONE OPERAND, THEN THAT OPERAND IS OUTPUT AND EXIT IS -. IF THE RESULTING EXPRESSION HAS NO OPERANDS, THEN INTE(O) IS OUTPUT AND EXIT IS -. OTHERWISE EXIT IS +.

NEGATION --- ACTION LIST CONTAINS R61, R4, R7, R11, AND R12.

R51. IF X IS ANY EXPRESSION, THEN -(^(...,X,...)) REDUCES TO v(...,~(X),...) AND -(v(...,X,...)) REDUCES TO ^(...,~(X),...). EACH NEWLY CREATED EXPRESSION, I.E. THE -(X), IS REDUCED VIA A RECURSIVE CALL OF THE SIMPLIFICATION EXECUTIVE FROM WITHIN THE ACTION. EXIT IS - WHEN THE ACTION IS APPLICABLE.

R4. IS X IS ANY EXPRESSION, THEN -(-X)) REDUCES TO X. EXIT IS - WHEN THE ACTION IS APPLICABLE.

R7. IF X AND Y ARE ANY EXPRESSIONS, THEN -(<(X,Y)) REDUCES TO <(Y,+(INTE(1),X)). THE OUTPUT EXPRESSION IS REDUCED VIA A RECURSIVE CALL OF THE SIMPLIFICATION EXECUTIVE FROM WITHIN
THE ACTION. EXIT IS - WHEN THE ACTION IS APPLICABLE.

R11. \( \neg(\text{SYMB}(\text{TRUE})) \) REDUCES TO \( \text{SYMB}(\text{FALSE}) \). EXIT IS - WHEN THE ACTION IS APPLICABLE.

R12. \( \neg(\text{SYMB}(\text{FALSE})) \) REDUCES TO \( \text{SYMB}(\text{TRUE}) \). EXIT IS - WHEN THE ACTION IS APPLICABLE.

CONJUNCTION --- ACTION LIST CONTAINS R80 AND R78.

R80. IF X IS ANY EXPRESSION, THEN \( \wedge(\ldots,X,\ldots,X,\ldots) \) REDUCES TO \( \wedge(\ldots,X,\ldots) \).

R78. \( \wedge(\text{SYMB}(\text{TRUE}),\text{SYMB}(\text{TRUE}),\ldots,\text{SYMB}(\text{TRUE})) \) REDUCES TO \( \text{SYMB} \) (TRUE), AND \( \wedge(\ldots,\text{SYMB}(\text{FALSE}),\ldots) \) REDUCES TO \( \text{SYMB}(\text{FALSE}) \). EXIT IS - WHEN THE ACTION IS APPLICABLE.

DISJUNCTION --- ACTION LIST CONTAINS R81 AND R79.

R81. IF X IS ANY EXPRESSION, THEN \( \vee(\ldots,X,\ldots,X,\ldots) \) REDUCES TO \( \vee(\ldots,X,\ldots) \).

R79. \( \vee(\text{SYMB}(\text{FALSE}),\text{SYMB}(\text{FALSE}),\ldots,\text{SYMB}(\text{FALSE})) \) REDUCES TO \( \text{SYMB}(\text{FALSE}) \), AND \( \vee(\ldots,\text{SYMB}(\text{TRUE}),\ldots) \) REDUCES TO \( \text{SYMB}(\text{TRUE}) \). EXIT IS - WHEN THE ACTION IS APPLICABLE.

\( \text{VEXCL} \) --- ACTION LIST CONTAINS R59 AND R60.

R59. IF X IS ANY EXPRESSION, THEN \( \text{EXCL}(\ldots,X,\ldots,X,\ldots) \) REDUCES TO \( \text{SYMB}(\text{FALSE}) \). EXIT IS - WHEN THE ACTION IS APPLICABLE.

R60. IF \( X_1,X_2,\ldots,X_N \) ARE EACH \( \text{INTEV} \) OR \( \text{SYMBV} \) EXPRESSIONS AND NO TWO \( X_i \) ARE IDENTICAL, THEN \( \text{EXCL}(X_1,X_2,\ldots,X_N) \) REDUCES TO \( \text{SYMB}(\text{TRUE}) \). EXIT IS - WHEN THE ACTION IS APPLICABLE.


R9. IF I AND J ARE INTEGERS SUCH THAT \( I<J \), THEN \( <(\text{INTE}(I),\text{INTE}(J)) \) REDUCES TO \( \text{SYMB}(\text{TRUE}) \). IF I AND J ARE INTEGERS SUCH THAT \( I<J \), THEN \( <(\text{INTE}(I),\text{INTE}(J)) \) REDUCES TO \( \text{SYMB}(\text{FALSE}) \). EXIT IS - WHEN THE ACTION IS APPLICABLE.
R51. IF X IS ANY EXPRESSION AND Y IS NOT AN VINT EW EXPRESSION, THEN <(X,Y) REDUCES TO <(+X,-Y),INTE(0)). THE EXPRESSION +X,-Y) IS REDUCED VIA A RECURSIVE CALL OF THE SIMPLIFICATION EXECUTIVE FROM WITHIN THE ACTION.

R53. IF I AND J ARE INTEGERS, THEN <(+INTE(I),...),INTE(J)) REDUCES TO <(+...),INTE(J-I)) AND <(INTE(J),+(INTE(I),...)) REDUCES TO <(INTE(J-I),+(...)). IF THE +(....) EXPRESSION HAS ONLY ONE OPERAND, THEN IT IS REPLACED BY ITS OPERAND.

R56. IF I AND J ARE INTEGERS AND X1,X2,...,XI ARE IDENTICAL EXPRESSIONS, THEN <(+X1,X2,...,XI),INTE(J)) REDUCES TO <(X1,INTE(K)) WHERE K IS THE SMALLEST INTEGER GREATER THAN OR EQUAL TO J/I; ALSO <(INTE(J),+(X1,X2,...,XI)) REDUCES TO <(INTE(N),X1) WHERE N IS THE LARGEST INTEGER LESS THAN OR EQUAL TO J/I.

R10. IF X IS ANY EXPRESSION AND I IS AN INTEGER, THEN <(-X),INTE(I)) REDUCES TO <(INTE(-I),X), AND <(INTE(I),-(X)) REDUCES TO <(X,INTE(-I)).

EQUAL --- ACTION LIST CONTAINS R3, R6, R17, R8, R16, R19, AND R50.

R3. IF X IS ANY EXPRESSION, THEN =<X,X) REDUCES TO SYMB(TRUE). EXIT IS - WHEN THE ACTION IS APPLICABLE.

R5 IF X AND Y REPRESENT DISTINCT IDENTIFIERS AND I AND J ARE DISTINCT INTEGERS, THEN =<SYMB(X),SYMB(I)), =<INTE(I), INTE(J)), =<INTE(I),SYMB(X)), OR =<SYMB(X),INTE(I)) REDUCE TO SYMB(FALSE). EXIT IS - WHEN THE ACTION IS APPLICABLE.

R17. IF X IS A VSYMB OR VINTE EXPRESSION AND Y IS ANY EXPRESSION, THEN =<X,Y) REDUCES TO =<Y,X).

R3 IF X IS A +, -, OR VSELEV EXPRESSION AND Y IS NOT AN VINTE EXPRESSION, THEN =<X,Y) REDUCES TO =<(+X,-Y),INTE(0)) AND =<Y,X) REDUCES TO =<(Y,-X),INTE(0)). THE EXPRESSIONS +<X,-Y) AND +(Y,-X)) ARE REDUCED VIA A RECURSIVE CALL OF THE SIMPLIFICATION EXECUTIVE FROM WITHIN THE ACTION.

R16. IF I AND J ARE INTEGERS, THEN <+(INTE(I),...),INTE(J)) REDUCES TO <(+...),INTE(J-I)). IF THE EXPRESSION +(....) HAS ONLY ONE OPERAND, THEN IT IS REPLACED BY ITS OPERAND.

R19. IF I, J, AND K ARE INTEGERS SUCH THAT I DIVIDES J EVENLY BUT
I does not divide \( k \) evenly and \( x_1, x_2, \ldots, x_I \) are identical expressions, then \( = (\cdot(x_1, x_2, \ldots, x_I), \text{int}(j)) \) reduces to \( = (x_1, \text{int}(j/i)) \) and \( = (\cdot(x_1, x_2, \ldots, x_I), \text{int}(k)) \) reduces to \( \text{symb} \text{(false)} \). If the reduction is to \( \text{symb} \text{(false)} \), then exit is -.

R50. If \( x \) is any expression and \( i \) is an integer, then \( = (-x), \text{int}(i) \) reduces to \( = (x, \text{int}(-i)) \).
APPENDIX II: ACTIONS FOR THE PHASE 3 CONSTRAINT PROCESSOR

PHASE 3 ACTION ROUTINES:

R40. IF X AND Y ARE ANY EXPRESSIONS, THEN = (X, Y) AND = (Y, X) IMPLY DELETION OF = (X, Y).

R30. IF X IS ANY EXPRESSION, C IS A \( \wedge \) SYMB\( \wedge \) OR \( \wedge \) INTE\( \wedge \) EXPRESSION, ONE CONSTRAINT IS = (X, C), AND THE OTHER CONSTRAINT CONTAINS X AS A SUBEXPRESSION, THEN DELETE THE SECOND CONSTRAINT AND FORM A NEW CONSTRAINT BY REPLACING ALL OCCURRENCES OF X IN THE SECOND CONSTRAINT BY C.

R34. IF X IS ANY EXPRESSION, THEN \( \neg \) (X) AND X IMPLY AN INCONSISTENCY.

R33. IF X AND Y ARE ANY EXPRESSIONS, THEN \( < \) (X, Y) AND \( < \) (Y, X) IMPLY AN INCONSISTENCY.

R35. IF X IS ANY EXPRESSION AND I AND J ARE INTEGERS SUCH THAT \( J \leq I + 1 \), THEN \( < \) (INTE (I), X) AND \( < \) (X, INTE (J)) IMPLY AN INCONSISTENCY.

R103. IF X IS ANY EXPRESSION AND I AND J ARE INTEGERS SUCH THAT \( I < J \), THEN \( < \) (X, I) AND \( < \) (X, J) IMPLY DELETION OF \( < \) (Y, J); ALSO, \( < \) (I, X) AND \( < \) (J, X) IMPLY DELETION OF \( < \) (I, X).

R109. IF X AND Y ARE ANY EXPRESSIONS, THEN = (X, Y) AND EXCL(\( \ldots \), \( \ldots \), \( \ldots \), \( \ldots \)) IMPLY AN INCONSISTENCY.

R110. IF X AND Y ARE ANY EXPRESSIONS, THEN \( \neg \) = (X, Y) AND EXCL(\( \ldots \), \( \ldots \), \( \ldots \), \( \ldots \)) IMPLY DELETION OF \( \neg \) = (X, Y).

R32. IF X AND Y ARE ANY EXPRESSIONS, THEN \( < \) (X, Y) AND \( \neg \) = (X, Y)) (OR \( \neg \) = (Y, X)) IMPLY DELETION OF \( \neg \) = (X, Y) (OR \( \neg \) = (Y, X)).

PHASE 3 ACTION INDEX:

\( \neg \) = \( \neg \) R40, R30.
\( \neg \rightarrow \) \( \neg \) R30.
\( \neg \rightarrow \) \( \neg \) R30, R34.
\( \neg \rightarrow \) \( \rightarrow \) \( \neg \) R30, R109.
\( \neg \rightarrow \) \( \neg \) R30.
\( \rightarrow \) \( \rightarrow \) \( \rightarrow \) R33, R103, R36.
\( \rightarrow \) \( \rightarrow \) \( \rightarrow \) R3, R110.
\( \rightarrow \) \( \rightarrow \) R32.
BIBLIOGRAPHY

AMAREL, S., 1968. THE REPRESENTATIONS OF PROBLEMS OF REASONING ABOUT ACTIONS, IN MACHINE INTELLIGENCE 3 (D. MICHIE, ED.), EDINBURGH UNIVERSITY PRESS.


BLACK, FISCHER, 1964. A DEDUCTIVE QUESTION ANSWERING SYSTEM, DOCTORAL DISSERTATION, DIVISION OF ENGINEERING AND APPLIED PHYSICS, HARVARD UNIVERSITY, CAMBRIDGE, MASS.

BOBROW, L. G., 1964. NATURAL LANGUAGE INPUT FOR A COMPUTER PROBLEM SOLVING SYSTEM, DOCTORAL DISSERTATION, MATHEMATICS DEPT., MASSACHUSETTS INSTITUTE OF TECHNOLOGY, CAMBRIDGE, MASS.


FILIPAK, A. S., 1942. 100 PUZZLES: HOW TO MAKE AND HOW TO SOLVE THEM, CARLETON AND CO., PP 20-21.


HADLEY, G., 1964. NONLINEAR AND DYNAMIC PROGRAMMING, ADDISON-WESLEY PUBLISHING COMPANY.

IBM, 1967. IBM SYSTEM/360 MODEL 67 FUNCTIONAL CHARACTERISTICS, IBM SYSTEMS REFERENCE LIBRARY, FORM 27-2719-C.

KOCHEN, F., 1960. EXPERIMENTAL STUDY OF HYPOTHESIS-FORMATION BY COMPUTER, PROCEEDINGS OF THE LONDON SYMPOSIUM ON INFORMATION THEORY, EULFFEFWORTHS.

MCCAFFTHY, J., 1959. PROGRAMS WITH COMMON SENSE, PPOC. SYMPOSIUM ON MECH. OF THOUGHT PROCESSES, HER MAJESTY'S STATIONERY OFFICE, LONDON, 75-84.

MCCAFFTHY, J., 1962. TOWARDS A MATHEMATICAL THEORY OF COMPUTATION, PROCEEDINGS IFIP, MUNICH.

MCCAFFTHY, J., 1963. SITUATIONS, ACTIONS, AND CAUSAL LAWS, STANFORD ARTIFICIAL INTELLIGENCE PROJECT MEMO NO. 2, JULY.

MOTT-SMITH, G., 1954. MATHEMATICAL PUZZLES FOR BEGINNERS AND ENTHUSIASTS, COVER PUBLICATIONS.


NEWELL, A. (ED.), 1961. INFORMATION PROCESSING LANGUAGE-V MANUAL, SECOND EDITION, PRENTICE HALL.

NEWELL, A., AND FERNST, G., 1965. THE SEARCH FOR GENERALITY, PROCEEDINGS IFIP CONGRESS, SPARTAN.

NEWELL, A., 1969. HEURISTIC PROGRAMMING: III-STRUCTURED PROBLEMS, PROGRESS IN OPERATIONS RESEARCH (J. ARONOFSKY, ED.), VOL. III, WILEY PUBLICATIONS.


RAFHAEL, B., 1964. A COMPUTER PROGRAM FOR SEMANTIC INFORMATION RETRIEVAL, D.C. THESIS, DEPARTMENT OF MATHEMATICS, MASSACHUSETTS INSTITUTE OF TECHNOLOGY, CAMBRIDGE, MASS.

SHANNON, C., 1956. PROGRAMMING A DIGITAL COMPUTER FOR PLAYING CHESS, PHILOSOPHY MAGAZINE, MARCH, VOL. 41, PP. 356-375.

TURING, A. M., 1953. AUTHOR OF FIRST CF CHAP. 25, PP. 298-295, CF FASTER THAN THOUGHT (B. V. FOWDEN, ED.), LONDON: PITMAN.
We describe an effort to design a heuristic problem solving program in which
the primary concerns are with the generality of the program's input language
and the effectiveness and generality of the program's problem solving methods.

To obtain the desired generality and ease of problem statement in an input
language, we propose extending a programming language to form a nondeterministic
language which is suitable for stating problems. The extensions preserve the
representational power and generality of the data and control structures of the
base programming language. We discuss the scope and limitations of nondeterministic
programming languages as input languages and compare them with the input languages
used by other problem solving programs.

The program was designed to accept problems stated in a particular nondeter-
ministic programming language and to deal effectively with the diversity of problems
expressible in the language. The program translates a nondeterministic procedure
into a heuristic search problem in which each object in the search space is itself
a constraint satisfaction problem. The program combines heuristic search methods
and constraint satisfaction methods to conduct the search and to solve the problems
defined by the objects in the space. We present discussions of the program's
problem solving methods with emphasis on their generality and extensibility. The
behavior of the program on a set of example problems is described and analyzed.