2

AD-A955 862

(When Data Entered)

ATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

| | | |
|---|---|---|
| 1. REPORT NUMBER<br>AFOSR-TR- 90-0157 | 2. GOVT ACCESSION NO. | 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE (and Subtitle)<br>THE STUDNT PRODUCTION SYSTEM<br>A Study of Encoding Knowledge in Production Systems | | 5. TYPE OF REPORT & PERIOD COVERED<br>Interim<br>6. PERFORMING ORG. REPORT NUMBER |
| 7. AUTHOR(s)<br>Michael D. Rychener | | 8. CONTRACT OR GRANT NUMBER(s)<br>F44620-73-C-0074 |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS<br>Carnegie-Mellon University<br>Computer Science Dept.<br>Pittsburgh, PA 15213 | | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS<br>61101D<br>AO 2466 |
| 11. CONTROLLING OFFICE NAME AND ADDRESS<br>Defense Advanced Research Project Agency<br>1400 Wilson Blvd.<br>Arlington, VA 22209 | | 12. REPORT DATE<br>October 1975<br>13. NUMBER OF PAGES<br>105 |
| 14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)<br>Air Force Office of Scientific Research (NM)<br>Bolling AFB, DC 20332 | | 15. SECURITY CLASS. (of this report)<br>UNCLASSIFIED<br>15a. DECLASSIFICATION/DOWNGRADING SCHEDULE |

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

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

18. SUPPLEMENTARY NOTES

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

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

see back side of page

# DISCLAIMER NOTICE

UNCLASSIFIED
Technical
Report
[illegible]

DEFENSE
TECHNICAL
INFORMATION
CENTER

DTIC Acquiring Information -
Imparting Knowledge

[illegible]
Alexandria, Virginia [illegible]

UNCLASSIFIED

**THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.**

DEPARTMENT
of
COMPUTER SCIENCE

Carnegie-Mellon University

# The Studnt Production System

## A Study of Encoding Knowledge in Production Systems

by Michael D. Rychener
October 1975

Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pa 15213

Abstract. This paper describes a production system implementation of Bobrow's STUDENT program. The main features of the new program, Studnt, are described. Contrasts between the two versions are pointed out. A discussion of the implementation brings out several properties of production systems, especially with regard to control.

Studnt is then used as an example of the embedding of knowledge in a production system. The knowledge in Studnt is expressed as 218 natural language statements of three types: task-oriented knowledge, implementation and programming techniques, and knowledge about production system control. Task-oriented knowledge is characterized by an abstract model with 16 statements, which can be organized as a problem space. A detailed example illustrates how the knowledge is mapped to the production rule form. The knowledge is largely at the problem space level, with about a fourth of the statements dealing with programming techniques, and a much smaller fraction dealing with production system control. The knowledge analysis brings out the importance of the explicitness of unordered production systems with respect to determining the knowledge encoded in each production. The model of knowledge acquisition suggested by the analysis indicates unique properties for production systems with respect to programming, debugging, and augmentation. The analysis gives rise to some measures along eight understanding-system dimensions. Comparisons with other research and consideration of the processes involved in the analysis point up the need for further work on this approach.

This research was supported in part by the Defense Advanced Research Projects Agency under Contract no. F44620-73-C-0074 and monitored by the Air Force Office of Scientific Research.

## TABLE OF CONTENTS

SECTION PAGE

APPENDIX PAGE

## A. Introduction

This paper is concerned with Studnt, a production system implementation of the STUDENT program of D. Bobrow (1964a, 1964b). The analysis of STUDENT grows out of a more general research program whose aim is to rationalize the field of artificial intelligence (AI). The purpose is to clarify the scientific issues involved in AI, to characterize and justify the methods, and to firm up the theoretical and conceptual basis of AI. It is hoped that this would give better direction to research, bring about better teaching and learning of AI, improve the quality of reporting of research, and in general make AI more productive. The approach is to try to extend some sound preliminary work (Newell, 1969) by looking at specific AI programs. Given any system, questions were to be asked along the lines of: "Where is the intelligence in it?", "How does its behavior come about?", "What are the methods it uses?", "Is there some measure of its effectiveness?", and "Can we measure the relative contribution of its parts?". These questions arise naturally in the context of AI programs whose basis is heuristic search, where analysis and experimentation can lead, in a straight-forward way, to satisfactory answers. For instance, in evaluating a chess heuristic like the sorting of capture moves according to the value of the captured piece, it is possible to test various versions of a chess program and contrast their behavior.[⊛] That kind of evaluation is in consonance with the scientific tradition of gathering knowledge by controlled experiments. It is not possible to carry over that approach to an analysis of STUDENT because apparently minor variations in STUDENT's structure can give rise to major deficiencies in its behavior, so major that comparisons lose their significance. Therefore, we take the approach of making explicit and analyzing the knowledge embodied in STUDENT, and in measuring the degree to which that knowledge is understood by STUDENT. Then we can go on to determine what parts of the knowledge represent methods, what parts contribute intelligence, and so on. This paper presents some initial progress, including some tentative measures, and puts forth a conceptual structure that may shape future work.

The goal of exploring the properties of production systems (PSs)[⊛⊛] as an AI language provides a second motivation. A PS program specifies everything in its behavior in terms of condition-action rules. The conditions all refer to a common Working Memory which is the complete dynamic knowledge state of the program, and actions are simply changes to that knowledge state. In practice, the numbers of conditions and actions are both in the range of half a dozen to a dozen. There are no control primitives as such, but rather control is achieved through explicit elements of the Working Memory. From this small collection of rather abstract properties, there are some features of PSs that we might look for in a PS program: uniformity and explicitness of expression of the knowledge content; flexibility and intelligence in the sense of doing a significant amount of condition-testing for each small sequence of actions; flexibility also in the sense of being able to respond to unexpected items in the knowledge state; and modularity of knowledge organization, following from the way knowledge is encoded in small, independent units. In addition to these attractive properties, there is evidence that a PS-like organization is prominent in human cognition (Newell and Simon, 1972). The task area of Studnt is hardly

[⊛] This is being done by James Gillogly, as part of a Ph.D. thesis, in preparation.

[⊛⊛] PS is used to abbreviate production system in this paper; PSs is its plural; P will be used to abbreviate production, plural Ps.

one that places demands on the language that will exercise all of those properties, but nevertheless we will get some preliminary data from examining the extent that STUDENT's structures and concepts have changed in order to be functional in a different programming environment.

The choice of STUDENT was based on personal preference, on the availability of a good description of the program, including a listing of the program in a rule-based language, and on simplicity and expected ease of implementation.

Input to STUDENT (the original) was a story problem expressed in a highly restricted subset of natural language. STUDENT converted that to a set of equations plus a set of unknowns to be solved for, and then solved the problem. It was able to apply optional transformations, consult a global store of "knowledge", and ask the user for more information, in case the set of equations derived from the input was insufficient for a solution. A typical problem is:

> "The price of a radio is 69.70 dollars. If this price is 15 per cent less than the marked price, find the marked price."

STUDENT's version of the equations and variables to be found can be expressed as:

```
(price of radio) = ( (69.70) X (dollars) )
(price of radio) = ( (.85) X (marked price) )
(solve-for (marked price))
```

STUDENT's answer is: the marked price is 82 dollars.

Studnt is designed to do only part of the above, namely, the translation from English-subset expressions into algebraic equations. Studnt thus includes the most interesting segments of STUDENT from the point of view of problem solving and natural language processing. In addition that portion of STUDENT was written in a readable PS-like language (Meteor), and the relevant parts of STUDENT were included in Bobrow's report (1964a), so that the present implementation follows the content of original rather closely. The omitted portions, except for the equation-solving process, seem to be straight-forward extensions of Studnt, while the equation-solver is a distinct piece of program and rather peripheral to the interesting natural language and problem-solving issues.

So, given a problem similar in form to those given to STUDENT, Studnt outputs: a set of equations; the set of variables in those equations as represented by the natural language text of the input; and a set of variables to be solved for. In addition, Studnt outputs the equivalences that it is assuming between certain phrases (which became variables) in the natural language text.

Section B contains a description of Studnt, with progressively more detail towards the end of the section. The material starting with Section B.4 is optional for the first reading. Section C discusses the knowledge content of Studnt, and investigates knowledge interactions in forming the Ps. Some of the appendices deal with details of the Studnt processing, while the others are relevant to the knowledge section, as will be explained below.

Studnt is implemented in Psnlst (PS analyst), a PS language specifically designed for AI applications. A PS is an unordered set of rules, Ps, specifying changes to a symbolic model of a situation, to be applied according to satisfaction of explicit conditions on that model. In Psnlst, condition- or left-hand-sides (LHSs) of Ps match an associative, unstructured Working Memory of data instances (items), each of which is a list headed by a predicate, followed by arguments. On matching, changes as specified by the action- or right-hand-sides (RHSs) are made to the Working Memory, either adding or deleting instances. The match distinguishes between new and old data, and Ps are selected for matching according to a stack regime whereby those relevant to the newest data are tried first, with older ones pushed down for later consideration. The stack is called :SMPX, stack memory for production examinations. The set of Ps is thus ordered dynamically, not statically, if indeed it can be considered to be ordered at all. The following is a typical P:

```
T1; "HOW OLD->WHAT" :: TFSCAN(X) & EQHOW(X) & LEFTOF(X,Y) & EQOLD(Y)
            & LEFTOF(Y,Z)
      => MODLEN(-1) & EQWHAT(X) & WORDEQ(X,'WHAT) & NOT WORDEQ(X,'HOW)
            & LEFTOF(X,Z) & NEGATE(ALL);
```

"T1" is the label, "HOW OLD->WHAT" is a comment string, and the condition (LHS) and action (RHS) are conjunctions separated by "=>". T1 is intended to recognize the sequence "HOW OLD" and change it to "WHAT", deleting and updating "LEFTOF" links. This brief description should be sufficient for the reader to follow the examples scattered throughout the text. Appendix A gives a more systematic explanation of Psnlst features and explains in detail the various characters that are output by the running interpreter.

## B. The Studnt Production System

### B.1. General overview

The main processing of Studnt is driven by a single left-to-right scan of the input, dividing it into smaller units called chunks, which are then parsed before continuing the scan. During this initial scan three things are done to provide information for the parsing process. First, simple string transformations are made, mapping the input to a form more acceptable to later processes, for instance, "twice" is converted to "two times". Second, dictionary tags are attached to key words, for instance, "times" is tagged as an operator of class "OP1". Third, the initial scan detects the operator, in the portion scanned, which has the highest "precedence", according to the parsing scheme to be described below. After the occurrence of a question word or phrase, the initial scan goes into FV mode (FV for find-variable). Each type of FV, as determined by the first word, has its own chunking cues, and each chunk becomes a variable, which requires no parsing.

The parsing of a chunk is based on a system of precedences, in such a way that the chunk is split at the leftmost operator of the set of those operators having the highest precedence in the chunk. The chunk is split into two chunks, and each of these is processed in the same way. The precedence system, for instance, assigns a high value to "is", the main equation operator, and lower values to "plus", "times", and "the sum of", respectively. That is, the higher-precedence operators are assumed to apply to higher levels of the resulting expression tree, for instance, "a times b plus c times d" is taken to mean "(a times b) plus (c times d)".

When a chunk can be split no further, it is taken to represent a variable. Thus, noun phrases are determined by their boundaries (operators and delimiters), and the only knowledge about internal structure consists of the features used in determining equivalence with previous noun phrases. Each variable is compared to each previously-determined variable. Two variables are the same if they have the same words in the same positions, with the following exceptions: a phrase which is the "head" of a previous phrase is taken to refer to the same object, for instance, "the number of fish" will match to a previous phrase "the number of fish in the pond"; "the" corresponding to "a" is taken as a match; and so on. The features used are independent of the meaning of the nouns used, and dependent on properties of structure and function words (pronouns, determiners). A variable containing "this" might be taken as referring back to some previous variable, in particular the "subject" of the previous sentence (for sentences of the form "xxx is equal to ...", where xxx contains no operators). Alternatively, "this" refers to a whole expression, as in "this product", provided the previous sentence had an operator as its main connective different from EQUAL.

After each variable has been examined, the pieces of the original sentence are put back together into a tree-structured expression according to labels that were formed as the chunks were split. That is, as each chunk is split, a marker is formed for each half of the chunk, with a pointer back to its parent; the halves become operands, the parent becomes the operator at the node of the tree. The label of the parent chunk in turn points to its parent, and so on. The tree is built from the bottom up until labels run out, and if the operator at the top of the tree is "EQUAL", it is noted as an equation.

The subdivision of FV (find-variable) chunks is quite distinct from the preceding. An FV chunk is simply a list of one or more FVs, delimited in special ways according to the initial words of the FV chunk. For instance "What are" is followed by two or more FVs separated by "and". As another example, "How many ... do ... have ?" is taken to mean "what is the number of ... ... have ?", that is, the FV starts out, "the number of". Each portion of an FV chunk delimited in these special ways is taken to refer to a variable of the problem, and a comparison is made to previous ones until a match is found.

When the end of the input is reached, unreadable internal representations are transformed into lists suitable for output. The natural-language text corresponding to each variable is collected into a list, and variables determined to be FVs are gathered into a single list.

## B.2. An example problem in detail

This subsection summarizes Studnt's processing on the example TEST2. This should give a good idea of how Studnt works in a general way; fine details of the actual Ps and data representations are given in later subsections.

The run begins by inserting the full representation of the text of the problem into the Working Memory (Figure B.1). The last insertion gives the external representation of the text..

```
INSERTING (ASCAN PB-1) (PROBLEM PB-1) (TGSCANFIN SB-1)
  (LEFTOF SB-1 A1-1) (EQA A1-1) (WORDEQ A1-1 A)
  (LEFTOF A1-1 F2-1) (EQFIRST F2-1) (WORDEQ F2-1 FIRST)
  (LEFTOF F2-1 N3-1) (EQNUMBER N3-1) (WORDEQ N3-1 NUMBER)
  (LEFTOF N3-1 P4-1) (EQPLUS P4-1) (WORDEQ P4-1 PLUS)
  (LEFTOF P4-1 #5-1) (EQ6 #5-1) (WORDEQ #5-1 6)
  (LEFTOF #5-1 I6-1) (EQIS I6-1) (WORDEQ I6-1 IS)
  (LEFTOF I6-1 E7-1) (EQEQUAL E7-1) (WORDEQ E7-1 EQUAL)
   . . .
  (LEFTOF S34-1 N35-1) (EQNUMBER N35-1) (WORDEQ N35-1 NUMBER)
  (LEFTOF N35-1 ?36-1) (EQ? ?36-1) (WORDEQ ?36-1 ?)
  (LEFTOF ?36-1 SE-1)
  (STRLENGTH 36) (ENDMARK SB-1) (ENDMARK SE-1)
  (TEXT
   (A FIRST NUMBER PLUS 6 IS EQUAL TO A SECOND NUMBER .
      TWICE THE FIRST NUMBER IS THREE TIMES ONE HALF OF THE SECOND NUMBER .
      WHAT ARE THE FIRST NUMBER AND THE SECOND NUMBER ?))
```

Figure B.1 Initial Working Memory contents for TEST2

The portion starting with the first LEFTOF and ending with the last LEFTOF is the internal representation of the text, which is the argument of TEXT. Each word of the text has associated with it a token, A1-1, F2-1, N3-1, etc. A token consists of the first letter of the word concatenated with the position of the word in the text, then "-" and a number which gives the number of tokens that have been generated from the identifier which precedes the "-" (the final number insures uniqueness for all such generated tokens). Relations are then attached to these tokens (the structure of token names is never used internally). LEFTOF gives relative positions of tokens in the string, while EQwww (for some word www) and WORDEQ relate the tokens back to the external representation. (Why two predicates are necessary for this is explained in Section B.4.) The two tokens SB-1 and SE-1 are ENDMARK's marking the left (beginning) and right (ending) ends of the string, respectively. The first insertion, (ASCAN PB-1), is a signal that the problem is to be checked for clues as to whether it is an age problem. This age-problem check must be done before everything else, because transformations and other processing depend on the result. PROBLEM gives the problem an internal name, which is very rarely used. STRLENGTH means "string length", and its value is used in making estimates of certain quantities having to do with the monitoring processes (I Ps), which will be explained in Section B.4. TGSCANFIN is the single most important predicate in the above list, since it initiates the scanning process, at token SB-1.

The first major piece of processing has to do with the text up to the first period. The following describes the essence of this processing, ignoring many of the finer details. The first segment is the chunk C-1: (A FIRST NUMBER PLUS 6 IS EQUAL TO A SECOND NUMBER). After the initial scan, PLUS is marked as an operator of class OP2, with precedence 7. The EQUAL TO is deleted by a transformation, and IS is assigned precedence 8. The highest precedence in C-1 is thus 8, and the chunk is split at the IS, to form CL-1: (A FIRST NUMBER PLUS 6) and CR-1: (A SECOND NUMBER). CL-1 and CR-1 are labelled so that when fully parsed the tree for the arithmetic expression can be re-built from the fragments. For instance, we have (LABELU C-1 1 TOP) and (LABELU CL-1 2 C-1); thus, CL-1 has a level-2 label, with parent node C-1. The U in LABELU stands for "unfinished".

A precedence scan is now done on CL-1 (picked by virtue of its being leftmost of the "unfinished" chunks, computed by a numerical priority; the effect of the numerical ordering is similar to that of a stack) and a split occurs at PLUS, which is the only thing in CL-1 which has a precedence value. In general, the precedence scan picks the element with highest precedence for the next split, and in case of ties picks the leftmost such. CL-1 becomes CL-2: (A FIRST NUMBER) and CR-2: (6). CL-2 undergoes the precedence scan, and the absence of any precedences indicates that it is a variable chunk. The variable identification process is done, and since no other variables have the same form, it is given a new token, VAR-1, as its expression (a chunk has associated with it an expression, which may be trivially a single VAR token). CR-2 similarly becomes VAR-2. In the process of giving the two chunks expressions, LABELU is changed to LABELF, F for "finished", and the presence of two "finished" chunks with the same "unfinished" parent node (CL-1) results in assigning CL-1 the expression formed from its operator, which was noted when it was split, and its two descendant nodes, namely (PLUS VAR-1 VAR-2). Having done this, control passes again to the precedence scan, which now examines CR-1; CR-1 was formed in the first split, but was "forgotten" while the left half of the split was being parsed. CR-1 has no precedences, and becomes VAR-3, after checking that it is not identical to any of the other VAR's. This prompts the construction of (EQUAL (PLUS VAR-1

VAR-2) VAR-3), since the two descendants of C-1 are now "finished". This expression is marked as an equation (ISEQN) by noting that it has EQUAL as its operator, and that its expression-tree level is 1. The first chunk is now complete, and the scan resumes, starting at TWICE.

The second main chunk is processed in a way similar to the first. Three new transformations are applied before it is parsed: TWICE becomes 2 TIMES, ONE HALF becomes 0.5, and the OF after the 0.5 becomes TIMES.

The third main chunk, starting at WHAT, is an FV chunk, since WHAT is recognized as a QWORD (question-word). The action on the third chunk involves splitting it at the AND, and processing the two halves as variables. The variables (A FIRST NUMBER) and (THE FIRST NUMBER) are recognized to be the same, differing only in A as opposed to THE, so that (THE FIRST NUMBER) is known to be VAR-1. Similarly, (THE SECOND NUMBER) is VAR-3.

The portion of the Working Memory that gives the final solution is in Figure B.2.

```
ISEQN (C-1 (EQUAL (PLUS VAR-1 VAR-2) VAR-3))
  (C-2 (EQUAL (TIMES VAR-4 VAR-1) (TIMES VAR-5 (TIMES VAR-6 VAR-3))))
HASREPR (VAR-1 (A FIRST NUMBER)) (VAR-2 (6)) (VAR-3 (A SECOND NUMBER))
  (VAR-4 (2)) (VAR-5 (THREE)) (VAR-6 (0.5))
FVLIST (PB-1 ((VAR-1 VAR-3)))
EQVARCHUNK (C-3 CL-2) (C-4 CR-1) (CR-4 CL-2) (CR-6 CR-1)
```

Figure B.2 Final output for TEST2

ISEQN denotes the two equations found; HASREPR gives external representations for each of the VAR's; and FVLIST gives the list of FVs. Instances of each predicate are ordered lexicographically by their first element. The EQVARCHUNK instances give which chunks are assumed to be equivalent. We see that two occurrences of VAR-1 (CL-2) are noted in addition to the first, and also two other occurrences of VAR-3 (CR-1). (The chunk names, C-1, etc., refer to actual text segments, whereas the VAR's are more abstract, and can be represented by several different C's.)

## B.3. Comparison with the original

One of the primary differences in the overall processing between Studnt and STUDENT is due to Studnt's being driven by the left-to-right scan. The Meteor language had built-in facilities for efficient scanning over arbitrary string segments to pick out patterns; Psnlst is more general, and must do the scan more deliberately. The original repeatedly applied its templates to the entire input string until no more valid applications could be made, thus imposing an order on template application as opposed to Studnt's order of examining text. This means, for instance, that sentence-boundary templates in STUDENT were all applied before, say, the breaking of sentences into equations was started. Studnt proceeds in contrary fashion, making full use of all information seen in the scan up to a boundary, before continuing beyond that boundary. This contrast is quite visible in the actual programs. A significant portion of STUDENT consisted of sets of rules, with individual rules in those sets consisting of processing plus a branch to the initial rule in the set. Exhaustion of one set of templates led to a branch to another set. The corresponding left-to-right sequencing is evident in Studnt's "S" group of Ps, which control applications of the various rule sets at each scan point.

A second major difference arises from the internal representation. STUDENT was written in a language specifically oriented towards processing data organized as one-dimensional lists. The underlying language for Studnt, Psnlst, is designed to require all such structure to be explicit rather that built-in, partially for the purpose of allowing examination of just how much use is made of the string structure of the input, and partially for the purpose of retaining generality.

This can be illustrated by comparing a specific rule from STUDENT:

```
(*      (HOW OLD)          (WHAT)                                          IDIOMS)
```

to the corresponding rule from Studnt:

```
T1; "HOW OLD->WHAT" :: TFSCAN(X) & EQHOW(X) & LEFTOF(X,Y) & EQOLD(Y)
          & LEFTOF(Y,Z)
     => MODLEN(-1) & EQWHAT(X) & WORDEQ(X,'WHAT) & NOT WORDEQ(X,'HOW)
          & LEFTOF(X,Z) & NEGATE(ALL);
```

In the former rule, there are four elements: the label of the rule (actually * is just a place-holder, with control passing implicitly from the previous rule); the left-hand-side; the right-hand-side; the "GOTO" field of the rule. Some rules have an optional action sequence between the third and last positions. Note that the Studnt P makes explicit the LEFTOF links and the updating necessary for the transformation, while this is implicit in the STUDENT rule. Also, the Studnt rule has a data signal TFSCAN instead of the combination of a label, which might be the target of a GOTO, and a GOTO field. Overall, STUDENT had about 290 rules, which included high-level control and output printing, whereas Studnt has about 260 Ps, so that the advantages of the specialized notation seems to result in compression in size of rules rather than changing the number of rules in the entire system.

Minor differences can be noted in some of the details of the processing. Not everything done by STUDENT was in the program as published; thus certain assumptions were made along the way that resulted in some differences in the final results. For instance, STUDENT used a plural convention, converting occurrences of singular forms to their plurals ("1 span" becomes "1 times spans") whereas Studnt converts plurals to singulars ("6 feet" is "6 times foot"). STUDENT deleted occurrences of "the" and "a", so that noun phrase comparisons have some automatic equivalences, while Studnt retains those words, and uses explicit Ps to encode the knowledge that the difference between "the" and "a" is non-essential. In this case, and perhaps others, Studnt is less general, since it doesn't have Ps to handle all of the cases implied by STUDENT's mechanism; this specificity seems desirable from the standpoint of analysis of just what knowledge is required for the task. Studnt doesn't check for error conditions; STUDENT recognized a few limited types of "errors" in the input problems. Overall, Studnt performs as well as

STUDENT on the test problems published in the original report (given the more modest definition of "solution"), so that there is good reason to assert close similarity in knowledge content of the two versions (see Appendix E for results on that set of tests).

The ways in which the control of the two programs differs can be illustrated by displaying the actual code for processing that results in parsing the input according to the operator precedence tags. First, the rules from STUDENT, with commentary enclosed in %'s:

```
(*       ($ ($1 / OP1) $)     ((FN CAR (*K 2)))
                              (/ (*S LEFT (*K 1)) (*S RIGHT (*K 3)))        OPTST)
         % this stacks the left operand onto LEFT, the right onto RIGHT %
         ...
(OPTST   ($1 $)               (1)                                           $)
         % the operator itself is used to determine branch target %
         ...
(TIMES   ($)                  ((*EN LEFT))                                  *)
(*       ($1)                                                               OFOK)
         % tests for nonempty, prepares to work on left operand %
         ...
(OFOK    ($)                  ((*K TIMES (FN OPFORM (*K 1))
                                (FN OPFORM (*N RIGHT))))                    END)
         % the recursive step: these rules are all part of OPFORM %
```

Studnt does the same thing by a loop for the precedence scan (P20-P29, P50), followed by the split into operator and operands (C25, C60), followed by the assembly (C70):

```
P20; "NEW HIGH PREC" :: PRECSCAN(C,X) & HIGHPREC(C,N,Y) & HASPREC(X,M)
             & SATISFIES2(M,N,'(GREATERP M N)) & LEFTOF(X,W)
             & NOT CHUNKENDR(X,C)
     => PRECSCAN(C,W) & HIGHPREC(C,M,X) & NEGATE(1,2);
P23; "PREC SCAN ON" :: PRECSCAN(C,X) & HIGHPREC(C,N,Y) & HASPREC(X,M)
             & NOT SATISFIES2(M,N,'(GREATERP M N)) & LEFTOF(X,W)
             & NOT CHUNKENDR(X,C)
     => PRECSCAN(C,W) & NEGATE(1);
P26; "PREC SCAN ON" :: PRECSCAN(C,X) & NOT( EXISTS(N) & HASPREC(X,N) )
             & LEFTOF(X,W) & NOT CHUNKENDR(X,C)
     => PRECSCAN(C,W) & NEGATE(1);
P27; "PREC SCAN DONE" :: PRECSCAN(C,X) & HIGHPREC(C,N,Y) & HASPREC(X,M)
             &SATISFIES2(M,N,'(GREATERP M N)) & CHUNKENDR(X,C)
     => HIGHPREC(C,M,X) & PRECSCAND(C) & NEGATE(1,2);
P28; "PREC SCAN DONE" :: PRECSCAN(C,X) & HIGHPREC(C,N,Y) & HASPREC(X,M)
             & NOT SATISFIES2(M,N,'(GREATERP M N)) & CHUNKENDR(X,C)
     => PRECSCAND(C) & NEGATE(1);
P29; "PREC SCAN DONE" :: PRECSCAN(C,X) & NOT( EXISTS(N) & HASPREC(X,N) )
             & CHUNKENDR(X,C)
     => PRECSCAND(C) & NEGATE(1);

P50; "HASOP1" :: PRECSCAND(C) & HIGHPREC(C,M,X) & SATISFIES(M,'(EQ M 5))
     => HASOP1(C,X) & NEGATE(2);
```

```
C25; "OP1 BRK" :: HASOP1(C,X) & WORDEQ(X,XW)
            => CSPLIT(C,X,X) & HASOP(C,XW) & NEGATE(1);

C60; "SPLIT CHUNK" :: CSPLIT(C,LOCL,LOCR) & LEFTOF(X1,LOCL) & LEFTOF(LOCR,X2)
            & LABELU(C,N,P) & MXCPRIOR(M)
        => EXISTS(CL,CR) & NEWPLOP(C) & RRENAME(X2,C,CR) & LRENAME(X1,C,CL)
            & LABELU(CL,N+1,C) & LABELU(CR,N+1,C) & HASCPRIOR(CL,M+2)
            & HASCPRIOR(CR,M+1) & MXCPRIOR(M+2) & CHUNKENDL(X2,CR)
            & CHUNKENDR(X1,CL) & NEGATE(1,2,3,5);

C70; "FINISH SEG" :: LABELU(C,N,P) & LABELF(C1,M,C) & LABELF(C2,M,C)
            & HASOP(C,X) & SATISFIES(P,P NEQ 'TOP) & HASCPRIOR(C1,PR1)
            & HASCPRIOR(C2,PR2) & SATISFIES2(PR1,PR2,PR1 ?*GREAT PR2)
            & SATISFIES2(M,N,'(EQUAL (?*DIF M N) 1))
            & HASEXPR(C1,Y) & HASEXPR(C2,Z)
        => HASEXPR(C,<X,Y,Z>) & LABELF(C,N,P) & NEGATE(1);
```

(For help in understanding those Ps, the reader might refer to Section B.5.) How Studnt encodes the choice of which chunk to do the precedence scan on (P10) is not shown here, but it suffices to note that the choice is based simply on a numerical priority (HASCPRIOR) assigned to the chunks. How STUDENT makes the same selection is implicit in the recursive calling of OPFORM illustrated above.

One further example illustrates the differences in the languages used to express the two versions. STUDENT uses the following:

```
REMEMBER ( ... (PEOPLE IS THE PLURAL OF PERSON) ... )
```

where there are many similar phrases as arguments to REMEMBER, to set up internal properties which are then used by the rule:

```
(WORDS ($1)              0 (/ (*Q SHELF (FN GETDICT 1 DICT)))          WORDS)
```

which cycles repeatedly over the entire problem string. Studnt's corresponding rule is:

```
D61; "PEOPLE PL" :: TGSCAN(X) & EQPEOPLE(X) => ISPLURAL(X,'PERSON) & NEGATE(ALL);
```

Thus STUDENT could be augmented by adding rules of a natural form, but the class of such forms was rather small, and the larger issue of significant augmentation could certainly not be encompassed by this mechanism. One of the aspects of the Studnt knowledge analysis below is an approach to the more general problem of augmentation.

### B.4. Description of the productions

Now we describe the Ps of Studnt in some detail, in groups according to their function, pointing out features of interest with respect to the use of PSs. Some of the descriptions include a typical P and a trace segment (starting at "!") showing its operation. In order to understand everything in full detail, the reader will need to refer to the meanings of the predicates, Section B.5, the program listing, Appendix B, and perhaps the cross-reference, Appendix C. The groups of Ps in this subsection are ordered by importance, which corresponds to their order in the program (though such order has no effect on program behavior). There are twelve groups: S (scan), T (transformations), D (dictionary tags), P (precedence tags), M (main verbs), C (chunking), R (renaming), V (variable identification), F (FV chunking), A (age problem), B (building output), and I (information monitoring). P names in Studnt are a single letter (the letter of the containing group) followed by one or two digits, e.g. S13, perhaps in rare cases followed by another letter, e.g. V33R.

S Ps: Scanning the problem string (14 Ps)

The S Ps make the primary scan of the input, resulting in the application of transformations, the addition of dictionary tags, the segmentation into sentences, and the determination of the highest operator precedence seen in each segment scanned. The important predicates are: LEFTOF, TFSCAN, TFSCANFIN, ISSCANCHUNK, TGSCAN, TGSCANFIN, TGSCANFIN2, TFASCAN, TFASCANFIN, HIGHPREC, HASPREC, FVSCAN, ISSCANFV. These Ps have the effect of sequencing the firing of other sets of Ps to accomplish the things mentioned. This sequencing is explicit, using two signals for each evoked process. For instance, TFSCAN evokes the transformation processing, and TFSCANFIN signals that the TFSCAN signal has been examined. These two signals are both asserted by S13 (and others), but TFSCANFIN follows TFSCAN in being asserted, and is therefore stacked in :SMPX until all the consequences of the TFSCAN have been examined. The signals for major processing are asserted as follows: TFSCAN (transformations, see T Ps), TFASCAN (age-problem transformations, called optionally, see A Ps), TGSCAN (dictionary tags, D Ps), and TGSCANFIN2 (leads either to precedence checks of S20-S30, or to FVSCAN, see F Ps). S20-S30 determine the leftmost position that has the highest precedence.

S40 is the key to segmentation of the input at the period delimiter. The PRECSCAND assertion in the RHS of S40 evokes the extensive parsing process on the chunk just scanned, passing control to the P Ps. S40 also contains the start of the scan of the next segment (TFSCAN and TFSCANFIN); these signals are stacked in :SMPX throughout the parsing. S70 notes that the end of the input is reached, and signals the answer-building process (B Ps).

A typical S P:

```
S13; "TF SCAN" :: TGSCANFIN2(X) & LEFTOF(X,Y) & NOT ISDELIM(X) & ISSCANCHUNK(C)
            & CHUNKLEN(L)
       => TFSCAN(Y) & TFSCANFIN(Y) & INCHUNK(X,C) & CHUNKLEN(L+1)
            & NEGATE(1,5) & NOT TGSCAN(X);
```

```
! 7.    S13-1   "TF SCAN"
USING (TGSCANFIN2 A1-1) (LEFTOF A1-1 F2-1) (ISSCANCHUNK C-1) (CHUNKLEN 1)
INSERTING (TFSCAN F2-1) (TFSCANFIN F2-1) (INCHUNK A1-1 C-1) (CHUNKLEN 2)
  (NOT (TGSCANFIN2 A1-1)) (NOT (CHUNKLEN 1)) (NOT (TGSCAN A1-1))
```

This P firing moves the initial scan pointer from A1-1 to F2-1, i.e., from "A" to "FIRST", in problem TEST2. C-1 is the current chunk. Transformations are invoked on F2-1, A1-1 is added to C-1, and the lengin of the chunk goes from 1 to 2. This is the seventh P firing in the process of solving TEST2.

T Ps: Transformations on the input string (38 Ps)

These Ps specify that certain sequences of tokens in the input are to be replaced by equivalent sequences, so that the parsing process can work with a standard form of input. Examples of transformations were mentioned in Section B.2. Some Ps achieve this by changing external names associated with tokens, while others assert new tokens and remove the old ones. In doing this, the LEFTOF links are maintained, sometimes requiring changes to the scan pointers that were set up originally by the S Ps. There are many uses of the macros STRINGEQ and STRINGINS; for an explanation of what these expand into, see the comment at the very beginning of the Studnt program listing, Appendix B.

External names of tokens are encoded in two ways, by EQwww and WORDEQ, as we saw in Section B.2. WORDEQ's could be used everywhere, without a need for the EQwww's, except that since WORDEQ has an instance for every input token, there would be much more searching during the matching process. On the other hand, WORDEQ is required to give a direct link from a token to its external name, for instance in comparing arbitrary phrases for identity.

The T Ps form a non-deterministic if-statement (COND). All of their conditions are keyed to the TFSCAN signal, and the checking of the conditions is done in a non-deterministic order. When a P succeeds in matching, the result is to delete the TFSCAN signal, thus disabling any further firings of other transformations. Another view would call these Ps a subroutine, control being passed by a data condition instead of in the conventional way. Other sets of Ps in Studnt also maintain control of processing in a coherent way, but use a larger set of signals to achieve communication.

T50-T52 are used (as a sort of subroutine) by several other Ps to properly rearrange the global scan pointers in case old tokens become inoperative as a result of replacement. The S Ps function as if nothing had happened.

Example:

```
T2; "IS EQUAL TO->IS" :: TFSCAN(X) & EQIS(X) & STRINGEQ('(EQUAL TO),X,Y)
       => MODLEN(-2) & LEFTOF(X,Y) & NEGATE(ALL,-2);

! 26.   T2-1    "IS EQUAL TO->IS"
USING (TFSCAN I6-1) (EQIS I6-1) (LEFTOF I6-1 E7-1) (EQEQUAL E7-1)
  (LEFTOF E7-1 T8-1) (EQTO T8-1) (LEFTOF T8-1 A9-1)
INSERTING (MODLEN -2) (LEFTOF I6-1 A9-1) (NOT (TFSCAN I6-1))
  (NOT (LEFTOF I6-1 E7-1)) (NOT (EQEQUAL E7-1)) (NOT (LEFTOF E7-1 T8-1))
  (NOT (EQTO T8-1)) (NOT (LEFTOF T8-1 A9-1))
```

"IS EQUAL TO" is transformed to "IS" by removing the two extra words, E7-1 and T8-1, and by fixing LEFTOF pointers to make I6-1 left of A9-1. The first insertion is a signal to the I Ps that a change in problem length has taken place.

D Ps: Dictionary tags (43 Ps)

The tags applied to word tokens are: ISOP2, ISOP1, ISOP0, ISVERB, ISPERSON, ISPRON (optionally, only in age problems), ISPOSSPRON (another optional one), ISPLURAL, ISSINGULAR, ISQWORD, and ISDELIM. These tags are applied in a control environment similar to the that for the T Ps.

P Ps: Precedence scanning and tagging (23 Ps)

P1-P9 are sensitive to the tags applied by the D Ps, adding precedence values for operators. P10-P29 form a precedence-scanning process that is called after chunks scanned by the S Ps are split. P10 and P15 determine which chunk to scan next, according to the explicit sequencing tag, HASCPRIOR. The unscanned chunk with highest value is chosen.

Actually P10 also notes the next-highest chunk, and re-inserts the ISCHUNK predicate for that chunk. This is necessary to be sure that P10 or P15 will be tried again after a precedence scan is completed, because ISCHUNK, as used in P10 and P15, actually means a new ISCHUNK, at least for the C0 one. Each time the match is done, though (even if it fails to succeed using a particular ISCHUNK as the new one), all new ISCHUNK's become old, and without the re-assertion, P10 or P15 would not be examined again, resulting in neglecting some ISCHUNK's. So, in P10, the next-highest chunk is re-asserted, making it new again, and stacking it in :SMPX behind other data which cause other processing to be done before coming back for more precedence scanning. P15 checks that no other unprocessed ISCHUNK's exist, so that no re-assertion is necessary.

P20-P29 make up a precedence-scanning loop, going from left to right in the chunk, with the result that the leftmost instance of the highest precedence is selected. PRECSCAN is the scanning signal, CHUNKENDL is used to start the scan at the left end, and HIGHPREC records the progress. The set of Ps is a loop because each new assertion of PRECSCAN results in examination of the elements of the set to determine the next action.

P30-P75 emit signals that are picked up by C, M, or V Ps, depending on the particular signal; so, after the precedence is determined, the chunk is split at an operator, transformed according to its verb structure, or taken as a variable chunk with no further splits possible.

Example:

```
P10; "START PREC SCAN" :: ISCHUNK(C0) & CHUNKENDL(X,C0) & HASCPRIOR(C0,M0)
              & NOT PRECSCAND(C0) & ISCHUNK(C1) & HASCPRIOR(C1,M1)
              & SATISFIES2(M0,M1,M0 ?*GREAT M1) & NOT PRECSCAND(C1)
              & NOT( EXISTS(C2,M2) & HASCPRIOR(C2,M2)
                    & SATISFIES2(M0,M2,M2 ?*GREAT M0) & NOT PRECSCAND(C2) )
              & NOT( EXISTS(C3,M3) & HASCPRIOR(C3,M3)
                    & SATISFIES3(M0,M1,M3,'(GREATERP M0 M3 M1))
                    & NOT PRECSCAND(C3) )
       => PRECSCAN(C0,X) & HIGHPREC(C0,0,X) & ISCHUNK(C1);
```

! 68. P10-1 "START PREC SCAN"

```
USING (ISCHUNK CL-1) (CHUNKENDL A1-1 CL-1) (HASCPRIOR CL-1 3) (ISCHUNK CR-1)
  (HASCPRIOR CR-1 2)
WARNING (CR-1) ALREADY UNDER ISCHUNK *+
INSERTING (PRECSCAN CL-1 A1-1) (HIGHPREC CL-1 0 A1-1) (ISCHUNK CR-1)
```

A precedence scan is initiated on C1-1 at position A1-1, its left end. (ISCHUNK CR-1) is re-asserted so that P10 will be examined again, after C1-1 is processed, to look at CR-1. P10 insures that C0, assigned to C1-1, is the chunk with highest priority, and that no chunk has priority between C0 and C1, assigned here to CR-1.

M Ps: Main verbs, Miscellaneous post-tag transformations (10 Ps)

M10-M55 split or re-arrange chunks according to the main verb. M10 handles the simple "is" case. The others are much more complex. For instance, M20 applies in situations such as "Tom has twice as many fish as Mary has guppies", transforming it to "The number of fish Tom has is twice the number of guppies Mary has".

M60-M75 are sensitive to outputs of D Ps, either un-doing their effects, or carrying them somewhat further, according to context not taken into account in the tagging. These actions could be incorporated into D's; their form is a carry-over from the original STUDENT, which did the tagging and transforming in such a way that assumptions about the contexts used in M60-M75 could not be made until after all of the transformations had been done. The left to right scan in Studnt removes that difficulty.

C Ps: Chunk splitting and re-combining (19 Ps)

C2-C55 act on the signals sent by P1-P9, by setting up to split chunks at the marked operators. The actual splitting and attendant bookkeeping is done by C60. C70-C78 put the chunks back together after they are parsed fully, with a separate P for each of three cases. C75 and C78 are concerned with saving referents of future "this" (this is only done for the highest level in the sentence, so that C70 handles other cases). C80-C85 handle bookkeeping for the "this" referents. C90 notes that a completed expression is an equation. The important predicates for this segment are: CSPLIT, URENAME, HASUOPCHUNK, ISUOPDUM, NEWREFEXPR, ISREFEXPR, ISEQN.

C15-C52 (except C25) are somewhat more complex than the other Ps. Their purpose is to control the parsing of unary operators (square, squared) in such a way that the single operands of the operators are parsed before further action is taken. This is as if parentheses were put around the operands. It is necessary to do this because the other operators in Studnt are binary, and expect a variable as argument. But in the case of, say, "two times the square of the number", the second operand of the "times" is the unary-operator expression. Thus the unary operators insert a dummy where the unary expression used to be, rename the unary expression as another chunk (using URENAME and Ps C20-C22), parse the unary expression, and signal that the dummy stands for the unary expression, so that it won't be treated as text when the ordinary processing gets to it (see V10).

```
C70; "FINISH SEG" :: LABELU(C,N,P) & LABELF(C1,M,C) & LABELF(C2,M,C)
                & HASOP(C,X) & SATISFIES(P,P NEQ 'TOP) & HASCPRIOR(C1,PR1)
                & HASCPRIOR(C2,PR2) & SATISFIES2(PR1,PR2,PR1 ?*GREAT PR2)
```

```
        & SATISFIES2(M,N,'(EQUAL (?*DIF M N) 1))
        & HASEXPR(C1,Y) & HASEXPR(C2,Z)
    => HASEXPR(C,<X,Y,Z>) & LABELF(C,N,P) & NEGATE(1);

! 112. C70-1   "FINISH SEG"
USING (LABELU CL-1 2 C-1) (LABELF CL-2 3 CL-1) (LABELF CR-2 3 CL-1)
  (HASOP CL-1 PLUS) (HASCPRIOR CL-2 5) (HASCPRIOR CR-2 4) (HASEXPR CL-2 VAR-1)
  (HASEXPR CR-2 VAR-2)
INSERTING (HASEXPR CL-1 (PLUS VAR-1 VAR-2)) (LABELF CL-1 2 C-1)
  (NOT (LABELU CL-1 2 C-1))
```

Two finished chunks, CL-2 and CR-2, which are variables VAR-1 and VAR-2, are formed into an expression using the operator PLUS of the parent chunk CL-1. CL-1 is marked finished (LABELF) and is ready to be formed into the expression of its parent C-1 (that won't occur, though, until the second operand, CR-1, is finished).

R Ps: Renaming chunks after splitting of a chunk (6 Ps)

R2-R4 rename a chunk going from right to left. R6-R9 rename a chunk going from left to right. R6-R9 additionally are able to name pieces of a sequence of text that were not previously in any chunk (R2 and R4 assume a previous chunk). New pieces of chunks as checked for by R6-R9 are added by Ps like M20. The important predicates are: INCHUNK, LEFTOF, CHUNKENDL, CHUNKENDR, LRENAME, RRENAME. Each group of R Ps is a loop, maintaining control structure through LRENAME and RRENAME instances. After completion of the renaming, the ISCHUNK signal is emitted, to be picked up by P Ps.

V Ps: Variable comparison, for equivalences (26 Ps)

V5-V37 perform a number of tests on new variable chunks (chunks with no operators), in order to determine if the chunk, or something very close to it, has been seen before. These tests are performed in a particular sequence, as controlled by instances of the predicates UNTESTED, THISTESTED, EQVARREMD, and EQCHUNKTEST. V5 emits the UNTESTED, after a check for a unary operator dummy; V10 handles the dummy case. V15-V21 check for "this" in the chunk, and resolve references accordingly. V23-V24 remove comparisons to variables that have already been proven equivalent to others (such comparisons would just be duplication of effort). V25 initiates comparison of the new variable to all previous variable chunks, except as just mentioned. The comparison is done by stepping through the variables to be compared, on the LEFTOF links, with either check for equality or check for correspondence according to several special equivalence conditions. These special conditions are checked by V31-V37, as follows: "the" = a previous "a"; "they" matches "the xxx", where xxx is an unspecified word, e.g. "the Russians"; "the" may be skipped; a singular form matches "the number of xxx", where xxx is the plural-form of the singular word (only for words that have been tagged by D's); "first number" = "one number" (the latter is in a new variable); "first number" = "one of the numbers" (latter is new); "second number" = "other number" (latter is new).

V40-V50 note that two variables are equivalent, when the comparison goes through the entire chunks being compared. V55 counts the variable chunks as they are compared to the new one, in a particular sequence to prevent the P match from finding multiple assignments; if it were allowed to find multiple ones, incrementing the count as kept by CHTESTED would be done only once, effectively, since each increment would use the value of CHTESTED before any of the multiple firings. Allowing multiple firings is a feature of Psnlst; it was used to advantage in V25, to find all comparisons to be made with a single match, but in V25, the order didn't matter, and no values depended on non-multiple firings.

The presence of V55 is actually not necessary, by analogy with a similar comparison process elsewhere in Studnt, A63-A69. The latter test makes better use of the implicit stacking mechanism of Psnlst; it was coded somewhat later in time than the V tests. V55 was left in because it seemed desirable to use it as an illustration of alternative methods of expression in Psnlst, and because it illustrates an approach applicable in more general situations, where stricter control is essential.

V60 notes that all tests are finished, and creates a new VAR token. V65-V90 are used to remove all testing signals from the Working Memory; this is useful in case one test succeeds before all the others are done, so that they need not be continued.

```
V30; "VAR =" :: EQCHUNKTEST(C1,C2,X,Y) & WORDEQ(X,XW) & WORDEQ(Y,XW)
            & LEFTOF(X,X2) & LEFTOF(Y,Y2) & NOT CHUNKENDR(X,C1)
            & NOT CHUNKENDR(Y,C2)
        => EQCHUNKTEST(C1,C2,X2,Y2) & NEGATE(1);

! 123.  V30-1   "VAR ="
USING (EQCHUNKTEST CR-1 CL-2 A9-1 A1-1) (WORDEQ A9-1 A) (WORDEQ A1-1 A)
  (LEFTOF A9-1 S10-1) (LEFTOF A1-1 F2-1)
INSERTING (EQCHUNKTEST CR-1 CL-2 S10-1 F2-1)
  (NOT (EQCHUNKTEST CR-1 CL-2 A9-1 A1-1))
```

This is an example of the variable comparison process. In this case the next positions to be tested will not be the same, since CR-1, "A SECOND NUMBER", is being matched to CL-2, "A FIRST NUMBER".

F Ps: FV scanning and segmentation (15 Ps)

The type of scanning and segmentation for FV chunks depends only upon the initial question-words. For instance, if a sentence starts with "What are", Studnt expects more than one variable, separated by "and". These expectations are set up by asserting instances of: RTANDQMGOING, RTQMGOING, RTDOGOING, RTDOESGOING, RTHAVEGOING, RTANDPERGOING. The scan is actually sequenced by the S Ps, using FVSCAN. In a couple of cases, more complicated transformations are done, for instance, F45 will change phrases like "How many fish does Mary have?" to "the number of fish Mary has". Example:

```
F5; "WHAT ARE FV" :: FVSCAN(X) & EQWHAT(X) & ISSCANFV(C) & CHUNKENDL(X,C)
            & LEFTOF(X,Y) & EQARE(Y) & LEFTOF(Y,Z)
        => CHUNKENDL(Z,C) & RTANDQMGOING(C) & NEGATE(1,4);

! 439.  F5-1    "WHAT ARE FV"
USING (FVSCAN W27-1) (EQWHAT W27-1) (ISSCANFV C-3) (CHUNKENDL W27-1 C-3)
  (LEFTOF W27-1 A28-1) (EQARE A28-1) (LEFTOF A28-1 T29-1)
INSERTING (CHUNKENDL T29-1 C-3) (RTANDQMGOING C-3) (NOT (FVSCAN W27-1))
```

(NOT (CHUNKENDL W27-1 C-3))

Here the beginning of an FV chunk is noted, T29-1, starting "THE FIRST NUMBER", keyed to "WHAT ARE". A signal is set up so that "AND" and "QMARK" are treated appropriately when encountered.

A Ps: Age-problem transformations (44 Ps)

The age heuristics in Studnt closely parallel those in STUDENT, so that the following description is somewhat cryptic; scanning the Ps should help to fill in the details. Most of the relevant predicates start with "AGE". A1-A3 detect clues to whether a problem is an age problem; the occurrence of any of the special words is conclusive evidence. A11-A12 delete superfluous phrases. A15-A20 translate the occurrences of verbs like "will be" into more suitable forms. A24-A28 note the occurrence of phrases that may be used later on to modify age variables that are not otherwise modified. A31-A35 translate age operators into arithmetic operators, for instance "age 5 years from now" becomes "age pluss 5" (pluss has a different precedence from plus). A38-A43 detect the need for an age operator, as first noted by A24-A28, collect that operator, and place it in the string after the current age variable. A50-A59 replace an occurrence of "their ages" by a list of all age variables seen so far, separated by "and". These AGEREF's are collected in the order seen, by using a numeric argument. Pointers to all age variables are collected as scanned, by A61-A69, which also do a comparison, so that several occurrences of the same age variable do not appear in the replacement for "their". A71-A75 replace the occurrence of a personal pronoun by the first age variable seen. A81-A85 do a similar thing for a possessive pronoun.

B Ps: Build up answers (6 Ps)

Several functions are performed in building answers: chunks that are FVs are collected into a list, replacing the chunk name with the variable it stands for (B1-B2); a check is made for an answer unit (as in, "How many spans ..."), by B3; and the external representation of problem variables is collected for output, by B5-B8. Note that the FVs are collected in a particular order, by using HASCPRIOR. B2 constitutes a single-production loop, continually firing until all the ISFV's have been collected onto the FVLIST.

B5 is also a single-production loop of sorts: the RHS specifies that BUILDREPR is to be done, followed by a re-assertion of an ANSWERBUILD2 instance, which causes B5 to be examined again for more possibilities, and so on until the variables to be represented are exhausted. In the variable-representation collection process started by the B5 BUILDREPR assertion, since several variables may be equivalent, and since those that are equivalent have the same expression but not necessarily the same string representation, HASCPRIOR sequencing is used, so that the first representation seen in the scan is used as the collected list (the second HASREPR argument).

I Ps: Information gathering (13 Ps)

These Ps are not part of Studnt proper. Rather they monitor Studnt's progress by counting operators, variables, equations, and FVs, and by estimating how many more of those are likely to be found, assuming the worst case. These counts and estimates are recorded in SPACESIZES instances. The information as recorded was at one time used to attempt to measure the contribution of each P-firing towards reducing the combinatorial possibilities of the final output of the process. Thus, as each piece of new information is added, more is known about the form of the output, in terms of a reduction in the number of a priori possibilities. On the basis of that reduction, the ultimate "value" of each P might be measured, with due account being taken of the fact that it depends on outputs of previous Ps, and so on.

X Ps: Examples for testing (27 Ps)

Each X P contains the initial data for an example, including signals to start the Studnt processing. These tests are in sets of three, so that during testing, only a small amount of storage is taken up by problem statements. The modules represented by the EXPR's were loaded separately, and after testing, deleted, before loading the next set. Each test uses the macro INITPROB to translate from a string representation into a sequence of predicates with arguments, for the internal representation. INITPROB is explained in a comment at the very beginning of Appendix B.

## B.5. Description of the predicates

In the following alphabetical listing of predicate descriptions, conventions on the types of arguments have been adopted to shorten the descriptions and to ease comprehension. Unfortunately, this typing is not done in exactly the same way in the body of the program (its value was not realized soon enough). Six argument types are distinguished, based on the first letter of the argument:

c: chunk; a chunk is a sequence of tokens linked by LEFTOF which forms a unit.
l: list structure.
n: number.
p: position in string; each position is represented by a token, for which various properties can apply.
w: word; the external name for a chunk element, e.g. "TIMES".
x: other, to be explained with specific uses.

Arguments that are multiply used within a predicate description are numbered. If numbers for different types correspond, then the arguments also correspond, for instance, (c1,c2,p1,p2) refers to two chunks, and two positions in those chunks, with p1 in c1, and p2 in c2.

The reader can refer to Appendix C to find names of Ps (Appendix B) that use these predicates.

AGECOMP(p1,p2) loop status for comparing age variables in an age problem to see if a new one is the same as one already seen; the tokens at p1 and p2 are to be compared next.
AGECOMPFIN(p) signal that an age variable comparison has been initiated, for a new variable starting at p; creates a new AGEREF if not removed by the AGECOMP loop.
AGECOMPREM(p) delete all AGECOMP signals, since the test has failed.

AGEOP(p,c) p starts an age operator for c; the operator may be used later in the chunk to modify an age variable that is otherwise unqualified.

AGEOPNEED(p1,p2,p3,l) collect the words of an AGEOP, as list l, with current collecting position p3; the result will fill in between p1 and p2.

AGEPOSSCOL(p1,l,p2) collect words starting at p1 into l; result is to replace the possessive pronoun at p2.

AGEPROB(x) x is an age problem; this enables special heuristic transformations and processing.

AGEPRONCOL(p1,l,p2) collect words starting at p1 into l; result is to replace the pronoun at p2.

AGEREF(p,n) p is the starting position of an age variable with priority n (lower means seen before); an age variable is any age problem variable which starts with a person.

AGEREFCHK(p) phrase starting at p is to be checked to see if it is a new distinct age variable (AGEREF).

AGEREFCNT(n) count AGEREF's, for assigning priorities to new ones.

ANSUNITCHK(x) check for creation of an ANSUNIT, in the process of answer-building for problem x.

ANSWERBUILD(x) signal that the answer-building process should begin for problem x.

ANSWERBUILD2(x) signal the check for initiation of the collection of the external representation of variables, in answer-building, problem x.

ASCAN(x) do preliminary check for keywords signifying an age problem; x is the current problem.

BUILDREPR(x) build up the external string representation for variable x.

CHTCOUNTED(c1,c2) in the variable-test counting process, marks c1 as having been counted with respect to tests on c2

CHTESTED(c,n) c has been tested with respect to n other chunks; initialized to 1 to include c itself.

CHUNKENDL(p,c) element at p is at the left end of c.

CHUNKENDR(p,c) element at p is at the right end of c.

CHUNKLEN(n) current length of current scan chunk is n; used in LPs.

CSPLIT(c,p1,p2) chunk c is to be split into two chunks, with p1 directly to the left of the operator phrase at the split, and p2 directly to the right.

DEFOPLIST(n,w) the n'th definite operator found in w.

DELAYEXPND(x) PSnlst primitive for delayed expansion of a PSMACRO; used here because of insertion of new, variable text during the problem runs.

ENDMARK(p) an end of the problem text string is at p (left or right end).

EQCHUNKTEST(c1,c2,p1,p2) test for equivalence between c1 and c2, which are assumed to be variables.

EQVARCHUNK(c1,c2) c1 and c2 represent the same variable.

EQVARREMD(c) signal that all EQVARCHUNK's have been removed from consideration in the variable comparisons.

EQwww(p) the word at p in the string is equal to "www".

FVLIST(x,l) l is a list of FVs for problem x.

FVSCAN(p) signal to initiate check for special FV transformations at p.

FVSCANEND(p,c) p in c marks the end of an FV; results in the set-up for another FV to follow, or in detection of the end of the input string.

HASCPRIOR(c,n) c has priority n; lower means seen first, if the chunk was created in the initial scan; otherwise a higher value is given to the left chunk than to the right, when a chunk is split in two; values from later splits are higher than for earlier ones.

HASEXPR(c,x) c has expression x; x is either a token referring to a variable, or a list structure for the expression.

HASIS(c,p) c has IS as highest precedence element, at p.

HASOP(c,w) c has operator with name w; this will be used in constructing the output expression.

HASOPm(c,p) c has OPm, for m = 0 1 2, as highest precedence element, at p.

HASPREC(p,n) p has precedence n.

HASREPR(x,l) x has external representation l; usually the list of words for the token x of a variable chunk.

HASSQUARE(c,p) c has highest-precedence operator SQUARE, at p.

HASSQUARED(c,p) c has highest-precedence operator SQUARED, at p.

HASUOPCHUNK(p,c) p is a unary operator dummy, set up to hold a position in c while the unary operator expression it represents is parsed; result will replace the dummy as an operand in c.

HASVERB(c,p) c has a verb as highest precedence element, at p.

HIGHPREC(c,n,p) the highest precedence for c is n, at p.

IFDELETED(x) signal that an IF has been deleted in the scan; x is a dummy argument.

INCHUNK(p,c) element at p is in c.

ISANSUNIT(w) w is the unit in which the answer is to be expressed; before the answer-building process, it is just a position in the string.

ISCHUNK(c) c is a (now) complete chunk; inserted after the entire chunk has been initially scanned, or after it has been renamed as a result of the splitting process.

ISDELIM(p) p is a delimiter.

ISEQN(c,x) c is an equation, with expression x.

ISFV(c) c is an FV.

ISIS(p) p is "is"; used to establish precedence value.

ISOPm(p) p is operator of class m, m = 0,1,2; used to establish precedence value.

ISPERSON(p) p is a person.

ISPLURAL(p,w) p is the plural form of w.

ISPOSSPRON(p) p is a possessive pronoun (only age problems).

ISPRON(p) p is a pronoun (only age problems).

ISQWORD(p) p is a question-word.

ISREFEXPR(c) c is a reference expression, ie., a candidate for a future "this"; c is either a sentence that isn't an equation or the subject of a sentence.

ISSCANCHUNK(c) c is currently being scanned; it is not an FV.

ISSCANFV(c) c is an FV, and is currently being scanned.

ISSINGULAR(p) p is the singular form of some word..

ISUOPDUM(p) p is a unary operator dummy, see HASUOPCHUNK.

ISVARCHUNK(c) c is a variable chunk, ie., no operators, a noun phrase; this is a signal for initiation of variable comparison processes.

ISVERB(p) p is a verb.

LABELF(c1,n,c2) c1 is labeled finished, expression-tree level n, parent c2.

LABELU(c1,n,c2) c1 is labeled unfinished, expression-tree level n, parent c2.

LEFTOF(p1,p2) p1 is directly to the left of p2.

LRENAME(p,c1,c2) c1 is renamed to c2, current position p, proceeding to the left from p.

MODLEN(n) modify the length of the string of the problem by n; used for estimating space sizes in I Ps.

MODLENC(x) x is a dummy argument; a chunk boundary has been reached; the string length used to compute worst-case space-sizes (I Ps) can be adjusted based on the length of the chunk just scanned.

MXCPRIOR(n) maximum chunk priority number is n; used to assign to each chunk a unique order number.

NEWDVAR(c) c is a new distinct variable; signal to I Ps.

NEWEQN(x) signals a new equation to I Ps.

NEWFV(c) c is a new FV; signal to I Ps.

NEWOP(x) signal that x is a new operator; for I Ps.

NEWPLOP(x) signal a newly-placed operator to the I Ps.

NEWPLVAR(c) c is a newly-placed variable; signal to I Ps.

NEWREFEXPR(c) signal a new reference expression, to become the ISREFEXPR.

NEWREFOP(w) signal that w is the operator of a reference expression, to I Ps.

NEWSIZE(x) signal that a new space-size vector needs to be computed; x is a dummy argument.

NUMVARCHUNKS(n) n distinct variable chunks are known.

PLACOPLIST(n,w) the n'th placed operator is w.

PRECSCAN(c,p) precedence scan is being done on c, current point p.

PRECSCAND(c) precedence scan has been done on chunk c; signal to note result and proceed accordingly, either to split chunk or test as variable.

PROBLEM(x) x is the name of the current problem.

PROBxxx(n...) where xxx is VARS, EQNS, OPS, or FVS; arguments are values contributing to space-size as noted in the comments accompanying I1 (see Appendix B); INDEF is an estimate based on string length of what is considered the worst case for the given quantity; ie, assumptions are made on lengths of entities giving rise to the largest expected count; DEF reflects actual count so far found; PLACED reflects that an operator or variable may be determined but its position in the output expression tree remains undetermined.

RRENAME(p,c1,c2) c1 is renamed to c2, current position p, proceeding to the right from p.

RTANDPERGOING(c) signal to apply FV transformations when "PERIOD" or "AND" is scanned, somewhere to the right of the current scan position; c is the current scan FV, limiting the scope of the signal.

RTANDQMGOING(c) similar to RTANDPERGOING, for "AND" or "QMARK".

RTDOESGOING(c) similar to RTANDPERGOING, for "DOES".

RTDOGOING(c) similar to RTANDPERGOING, for "DO".

RTHAVEGOING(c) similar to RTANDPERGOING, for "HAVE".

RTQMGOING(c) similar to RTANDPERGOING, for "QMARK".

SPACESIZEN(n) the number of space-size vectors.

SPACESIZES(n,l) l is the n'th space-size vector; components correspond to arguments for all of the PROBxxx's.

STRINGEQ macro for generating strings of EQwww's, LEFTOF's, etc. - see comment in program listing

STRINGINS macro for generating strings of EQwww's, LEFTOF's, etc. - see comment in program listing.

STRLENGTH(n) the length of the input string remaining to be scanned.

TANDDIFF(c) transform "AND" in c to "MINUSS", since the difference operator has preceded it.

TANDSUM(c) transform "AND" in c to "PLUSS", since the SUM operator has been seen.

TBYIS(c) transform "BY" to "IS", as required by "EXCEEDS".

TFASCAN(p) signal to check for special age-problem transformations.

TFASCANFIN(p) signal completion of TFASCAN at p.

TFOUT(p1,p2) rearrange the TFSCAN pointers that used to be at p1, to be at p2; necessary in some transformations that actually re-order the string.

TFOUTDELAY(p1,p2) do a TFOUT on p1 and what becomes to the left of p2, after insertion of generated, variable text.

TFOUTLEN(p1,p2,n) TFOUT with a string length adjustment of n.

TFSCAN(p) signal to initiate check for string transformations at p.

TFSCANFIN(p) signal completion of TFSCAN at p, ready for next step in the scan process.

TGSCAN(p) signal to initiate check for dictionary tags at p.

TGSCANFIN(p) done with TGSCAN at p, record precedences or do FVSCAN; also a special signal to initiate the scan to begin the problem.

TGSCANFIN2(p) completion of initial scan processing at p, ready to move scan pointer.

THEIRCOLL(p1,p2,p3,p4,l) collect an age variable starting at p3, current collection position p4, list of text l, to be inserted along with other variables between p1 and p2 when collected.

THEIRCOLLD(p1,p2) age reference starting at p1 has been collected, for "THEIR" which is to be replaced at p2.

THEIRREF(p1,p2) a signal to collect a list of all ages seen so far, which are referred to by "THEIR", and put them between p1 and p2 when collected.

THEIRREFL(l) a list of all text collected so far for a "THEIR" replacement; each variable is collected separately and then added to this list.

THISTESTED(c) the variable test for "THIS" has been done for c; signals the initiation of the match of c against other variable chunks.

UNTESTED(c) c is not tested with respect to equivalences with other variables; signals for the first of a series of tests to be started.

URENAME(c1,c2,p3,p4,p5) c1, which is the operand for a unary operator, is to be renamed to be c2; renaming is currently at p3, to be terminated at p4; on termination, the chunk is to be split at p5.

VARCHCOUNT(c1,c2) signals failure of equivalence tests of c1 with respect to c2; chunks are counted after being tested.

VARCLEANUP(c) clean up assertions having to do with the testing of c, since the result is known.

WCOLLECT(c,x,p) collect words for c, with expression x, at p.

WORDEQ(p,w) the word at p is w.
WORDINS macro for generating EQwww and WORDEQ for a string position - see comment in program listing.

## B.6. Conclusions on the implementation

This subsection considers the following aspects: validation, program control, representation, and efficiency. First, in order to verify that Studnt is close to the original, Appendix E gives the results of test runs on 27 problems as given in the original publication. All of Studnt's answers are acceptable approximations to the solutions produced by STUDENT. These tests used all of the Ps of Studnt except: S65, T3, T6, T7, T19, T20, D1, D9, D13, D14, D65, D67, D75, D87, P8, P28, P65, M30, M50, C5, C50, C52, V21, A3, A15 (that is, 25 out of about 260). There is no essential difference between these Ps and Ps that were actually used for the tests, so that this deficiency is not serious.

Programs written in Psnlst must use data signals to provide control, as is the case in all PSs. Several features of Psnlst are useful in coordinating control signals. The main one is its stack memory, :SMPX, which is a temporary memory that effectively orders new elements of the Working Memory by their recency of assertion. Ps are selected for firing on the basis of this recency order, with those using the most recent data selected first, and with others pushed down in the stack until all the consequences of the newer data have been considered. The recency order is specified by the left-to-right order in RHSs of Ps, such that the left-most assertion is considered to be the most recent. If a data instance is re-asserted at some time after its initial assertion, it is given a higher position in the recency order, corresponding to its most recent assertion. This re-assertion is analogous to data rehearsal in other systems. Another Psnlst feature is that when a P is selected for matching, it may fire more than once, as opposed to firing once, allowing other Ps to be examined relative to the new data from that firing, and then returning to consider other possible matches that were available at the time of the original match. That is, all possible firings occur, in arbitrary order, before proceeding. Thus a set of Ps representing steps in some process can be working on more than one input element at a time, with multiple firings giving the appearance of parallel sequencing on the inputs.

In Studnt, control passes in various flexible ways between: S Ps and T, A, D, and F Ps; P and C, M, and V; C and R; M and R; R and P. The I Ps are evoked by most other groups. Appendix D gives a picture of the changes in control. The recursive nature of the parsing process, that is, the maintenance of the tree structure of the chunks, is encoded in the labels attached to chunks as they are split. Strict control sequencing is exhibited in the initial scan processing (S Ps), in the splitting of chunks (P10), in the variable comparisons (V Ps), and in the answer-building (B5). That is, the S, V and B Ps use specific signals to perform definite sequences of steps in fixed orders. The chunk-splitting process orders the chunks by attaching to each a numerical priority, and then processing according to that, resulting in the appearance of a stacking mechanism. The sequencing of the main scan, with control passing from S to (and from) T, A, D, and F Ps makes use of the stacking mechanism of :SMPX to order the consideration of process initiation and completion signals, which are emitted simultaneously by S Ps. That is, an S P emits both an initiation signal and a completion signal, with the initiation signal processed immediately and the other stacked in :SMPX for consideration after everything relating to the initiation signal has been completed. Many looping processes were noted : P20-P29, C20-C22, two in the R's, V5-V60, several in the A's, and two in the B's. A loop can easily maintain tight control by using a special signal which is asserted first in its actions, and which is only used by other Ps in the same looping process. The mechanism of re-asserting data to cause re-examination at some later point is used twice, in P10 and in B5. Multiple firing of Ps is used to advantage in three places, V25, A63, and A67, and special care is taken to prevent it in V55. In V25, for instance, a new variable is compared to all previous ones, with the set of previous ones considered all at once instead of serially. In summary, we see that in an environment without conventional control primitives it is straight-forward to achieve a variety of flexible control facilities.

The unstructured Working Memory of Psnlst is intimately connected with Studnt in two ways. The number of items in the memory is much larger than is efficiently stored in the linear Working Memory of other PSs. The range of Working Memory size for the Test2 example is from 115 to 321 items (these are initial and final figures, since no intermediate values are known, but no significant differences are expected for more accurate monitoring). The final memory size for Test16, the biggest test, is 765. The :SMPX mechanism narrows the focus of attention to a small portion of this mass, but even :SMPX becomes relatively large. For instance, the maximum number of :SMPX entries for Test2 is 126, but this is probably much larger than the number of distinct memory items that are referred to, since a data item occurs in many entries. Very little effort was made to limit the memory size, since the interpreter is capable of handling such magnitudes efficiently. Thus, these figures should not be taken as representative. The second effect of the Working Memory is that it is more general and more cumbersome than the special string representation used in STUDENT, but the benefit of making everything more explicit counteracts that minor difficulty, as we see in Section C.

The execution times of the tests given in Appendix E are in the range from 2 minutes to 20 minutes, with the average around 5.6 minutes (on a PDP-10 computer). This is within an order of magnitude of what would be considered reasonable times for these tasks as performed by humans. One might expect a computer with the limited knowledge that STUDENT has to do an order of magnitude better than that, so that PSs seem not particularly speedy⊗. Two things might easily make this order of improvement: more efficient implementation of the interpreter, and some way of compiling Ps (they're run interpretively at present). Also, the efficiency limitation may not be as serious as it appears, because one might argue that as more knowledge is added, little is added to total run time, since the number of applications of Ps in doing a particular task would not necessarily go up significantly. This assumes that not much is added to the time required for selection of the next P to fire. This is reasonable based on limited experience so far, which indicates that the ratio of examinations to firings is fairly low. (Humans probably have no problem with huge amounts of knowledge because of some parallelism in the recognition-selection process.) It also may be that new knowledge would interact only slightly with existing knowledge, so that there would be little interference with the

⊗ These times are in the right range for humans; the only STUDENT figure is that it took less than a minute (on a 7094) to do the age problem TEST6, which Studnt does in about 7.5 minutes, about a factor of 20-30 slower.

selection processes. That is, things that are relevant to present Ps would only rarely be relevant to new ones[⊗]. Memory usage is on the average about 95K 36-bit words. About 35K of that is devoted to the Lisp and Psnlst interpreters.

[⊗] This is similar to the problem space closure concept in Newell and Simon (1972), chapter 14, pages 819-820.

## C. The Knowledge in Studnt

The primary results presented in this section are based on viewing Studnt as the result of a knowledge encoding process. Philosophically this view is similar to McCarthy's Advice Taker proposal (1958), which laid out a plan for a general program that could modify its knowledge and its internal working procedures in accordance with advice given externally. The details of McCarthy's proposal were expressed with reference to a systematization of common sense knowledge as declarative statements in predicate logic, whereas the present approach expresses knowledge informally in unrestricted natural language and has a PS program as its target. That is, Studnt is analyzed as if it were the result of the assimilation of a large number of knowledge statements (KSs) in natural language. These KSs are shown to interact with each other to form the encoding of the knowledge as a PS.

The general strategy taken here is appropriate when viewed in the framework of a knowledge acquisition approach to AI. This general approach consists of several steps: a precise formulation of the knowledge that it is necessary or desirable for an AI program to have; a suitable programming language, interpretable by a computer, for the ultimate expression of knowledge as procedures and data; and some way to bridge the gap between the external representation and the internal (procedures and data) representation of the knowledge. This is to be contrasted with a knowledge generation approach, which I believe is implicit in approaches using mechanical theorem-proving techniques, perhaps inspired by McCarthy's Advice Taker. Knowledge generation takes knowledge in the form of axioms and operates on it according to inference rules, in the hope that knowledge sufficient to produce intelligent behavior will result. A generation approach does not distinguish the three steps above, in part because the internal and external representations are the same; also it is not concerned with exhibiting a full body of knowledge, but rather with finding an adequate basis for generation. Since the generation approach has not yet been successful, the present approach is proposed as an alternative. Since it is a first approximation, some aspects have been alluded to, illustrated, and circumscribed, but it remains informally (and vaguely) expressed. Expressing the knowledge precisely in any language (natural or artificial) is no small endeavor, and it is an activity that has not been carried out at the present scale by any previous work. The use of unrestricted natural language in the present work will be justified below (Section C.11).

At present, a computer program for the knowledge encoding process does not exist, although no insurmountable difficulties in constructing such a program can be foreseen. Rather, the knowledge has been obtained by an analysis (also not computerized) that represents a dual of encoding knowledge, namely, by a knowledge extraction process. The extraction is based on the meanings of the predicates that compose Studnt's Ps. Although the KSs were obtained analytically by an extraction process, it has seemed most natural to express them as if for use in encoding. Of course, Studnt is the result of an encoding process, but there is no basis for saying what the author had in mind during that original encoding, since accurate records were not kept.

The KSs fall quite readily into three major classes, which will be referred to as the N class, the Q class, and the Z class. The N-class statements (Ns) contain all of the task-oriented knowledge, for instance, knowledge about how arithmetic expressions are represented in natural language, how to recognize a specification of which variable is to be solved for, how to transform idioms, and so on. Most of the description of Studnt in the preceding section is at this level, loosely speaking. To organize this knowledge, we will use and augment slightly the concept of problem space (Newell and Simon, 1972, chapters 3 and 14), and we will refer to N statements as being at the problem space level.

Q-class statements (Qs) deal with implementation knowledge. These define terms used at the problem space level and provide a collection of programming techniques suitable for the requirements of the problem space. The Qs are stated in a sufficiently general way to be useful in conjunction with other problem domains than Studnt's domain and with other programming languages besides Psnlst.

The Z class of statements (Zs) deal with Psnlst control constructs, namely the special control features of Psnlst that affect the actual form of the Ps. The present analysis neglects other Psnlst features such as syntax and the properties of P conditions and actions; this level is suppressed because of its straight-forward, routine nature.

In addition to the three classes of KSs that comprise the abstract content of actual Ps, a fourth, concrete component is central to the analysis: the predicates, which are the problem-specific programming constructs. The knowledge extraction process is entirely dependent on the predicates' meanings (see the preceding section) for forming the KSs. The knowledge encoding process as presently formulated takes the predicates as given, and uses them at the appropriate (near-final) step in building the Ps. The predicates are the basic expressive primitive for all the KSs, so that their meanings span the three classes (N, Q, and Z).

The division of KSs into Ns, Qs, and Zs raises some interesting questions relating to what kinds of KSs might be necessary to augment Studnt's capabilities and relating to what might happen to the contents of each class as shifts to other programming languages, other task domains, and so on, are considered. But the division has also led to the hypothesization of a more general model of knowledge acquisition. The model puts the N, Q, and Z components into a larger framework, and indicates the location of some interesting topics for further work. It is used to display the interdependencies of those three classes, it makes more explicit what other knowledge is needed to complete the knowledge encoding process, and it allows questions about the origins of the Ns, Qs, and Zs to be posed. In particular there are interesting questions relating to the formation of the problem space that is the basis of Studnt. Finally, the model of knowledge acquisition makes contact with work by other researchers.

This section commences by presenting a model that can be used to give an overview of the Ns; the model describes the knowledge at the problem space level abstractly, and provides a basis for determining the relationships of various subsets of KSs. A definition of problem space is included in that discussion. Section C.2 goes through the knowledge encoding process for a particular P, illustrating how KSs interact and how contact with Studnt predicates is made. The interactions of KSs in forming a selection of other Ps is given in Section C.3, illustrating the uniformity of the encoding process over all of Studnt, and raising the question of "bugs" that became evident. The encoding process is summarized in Section C.4. We then shift the focus to the division into Ns, Qs, and Zs, giving abstract characterizations for the Qs and Zs to parallel the model given in Section

C.1; other aspects of the division are discussed at the same time. Section C.6 returns to the topic of knowledge extraction, the preceding subsections having laid a foundation for the necessary details. The more global view provided by the hypothesized knowledge acquisition model is elaborated in Section C.7. The last four subsections, Section C.8 through Section C.11, give conclusions, comparisons to other approaches, considerations with respect to understanding systems, and foreseeable problems in extending this work.

## C.1. Characterizing the content of the knowledge statements

The Ns are the class of KSs that deal with the knowledge in Studnt at the problem space level, namely knowledge about the task environment and how to deal with it (problem space is defined more precisely later in this subsection). In other words the Ns are a mixture of process-independent facts about the domain of algebra word problems and of knowledge about specific methods and control sequences that can be used to coordinate the application of the domain facts to produce appropriate problem-solving behavior. They are a mixture because they are what is immediately extractable from the Studnt Ps. As we will see below in discussing the model of knowledge acquisition (Section C.7), the consideration of pure task environment knowledge is one level removed from the problem space level, and in any case the problem space level cannot be bypassed, as that model is presently envisioned.

To provide an overview of the Ns and to establish a vocabulary of elements and relations, we propose a model, in the following sense. A model is a coherent body of objects and relations that represents some more complex structure, in such a way that manipulations (relations) on elements of the model correspond to manipulations (relations) on elements in the modelled structure. A model generally abstracts, suppressing some elements and relations and thus emphasizing others. In this sense a flowchart is a model of the control flow of a process.

The model of the Ns gives a global overview, grouping the Ns according to their more global function. For instance, key terms in the model (for instance, "chunk") are defined at some point, have relations to other terms, are manipulated or transformed, and so on, in ways that are clearly specified in the model. For more detail, the model provides pointers into the actual subsets of Ns. The presentation of the model at this point should help the reader to place the Ns that occur in the following subsections in perspective; the model is also essential to the identification of this level as the problem space level. The model is central to the knowledge encoding and knowledge extraction processes, but in ways that are difficult to pinpoint given the informal stage of the present analysis. That is, the use of such a model was evident at many places while the analysis was being done, but a clear picture of its use did not emerge; it probably will not do so until the processes are automated. We will discuss this further below.

The model of the Ns consists of statements a. through p. below. The objects in square brackets, such as [NS6-NS10, NS13], are sets of KSs that are elaborations of the associated model statement. The KSs are listed in full in Appendix F, and they will be discussed further in the subsections following this.

a. Input: a sequence of "words", each occupying one "position".

b. Output: a set of "equations" composed of "expressions" consisting of algebraic variables (domain: real numbers), real constants, and common arithmetic operators; a list of specific variables whose values in the solution of the set of equations is sought, with an optional "answer unit" in terms of which the answer is to be expressed; a set of assumed equivalences between sequences of words that stand for algebraic variables. [NB1-NB33].

c. For every sequence of words there is a desired (canonical) form, to which the sequence is transformed. [NT1-NT32, NM9-NM11].

d. A word may belong to one of several classes of words; other operations that depend on the word may use its class membership properties. [ND1-ND14, NM13].

e. The operations of transforming the input sequence and assigning words to classes are correct only if done in particular order relative to each other and within the word sequence; this sequencing is achieved by the "initial scan". [NS1-NS5, NS11, NS12, NS14, NP2, NC15].

f. The sequences are broken into "chunks" according to membership of words in particular word classes, and according to interrelationships between the words in these classes.

g. The first subdivision into chunks is based on membership of boundary words in a set of classes distinct from the classes that determine further subdivisions. [NS6-NS10, NS13].

h. Further subdivision of the chunks is conditional on certain class memberships, i.e., there are two ways of proceeding from the first subdivision. [NS4, NS5].

i. Under the first kind of further subdivision, the chunks are subdivided according to the properties of words of the "operator" class, and according to relative positions of these, as determined by a "scan", with each resulting chunk associated with the operator which formed its boundary as an "operand"; when a chunk is subdivided, the chunk membership property of the operand parts is changed by "renaming". [NP1, NP3, NM1, NM8, NC1-NC10, NC15, NR1, NR2].

j. One class of words requires a chunk to be rearranged in specific ways before it can be subdivided into variables and operators; i.e., "verbs". [NM2-NM7, NM12].

k. The resulting chunks and operators are then arranged as expressions in a tree structure (the tree structure is thus also determined by class memberships of its operators); such a tree structure with the operator "EQUAL" at its top node is an equation. [NC11, NC12, NC17].

l. A chunk that can be subdivided no further is termed a "variable"; variables which have similar word-sequence structure are assumed to refer to the same algebraic variable; similarity is determined by a set of rules; a variable may also refer to some previous expression. [NP4, NC14, NC16, NV1-NV18].

m. The second type of subdivision is determined in ways specific to particular word configurations; its result is the second output component, i.e., the list of variables to be solved for, termed "FVs". [NF1-NF9].

n. An input sequence that is recognizably of a particular class, "age problem", undergoes special transformations in addition to those normally applied in reaching the desired (canonical) form for sequences of words [NA1-NA11, NA13].

o. In an age problem, certain segments of text may be copied from one position to another, dependent on class memberships or on the presence or absence of particular word sequences. [NA12, NA14-NA17].

p. The result of certain of the above operations is that certain estimates of the size of the space of possible outputs can be made or adjusted. [NI1-NI10].

The concept of *problem space* arose out of the need to describe the space in which human problem solving activities take place (Newell and Simon, 1972, p. 59). In particular, it is essential to be able to describe the *possibilities for the behavior*, rather than being limited to describing only the actual behavior. As originally formulated (Newell and Simon, 1972, pp. 810-811) a problem space has five components: (1) a set of *elements*, each representing a state of knowledge about a task; (2) a set of *operators* that produce new elements from existing ones; (3) the *initial element*; (4) the *desired element* or set of elements, to be reached from the initial element by applying operators; (5) the *total knowledge* available, which ranges from temporary dynamic information to long-term reference information. This can be seen to be similar to a general formulation of the heuristic search method (see, for instance, Newell, 1969), but there are differences. In human problem solving, a set of invariant features that are restrictive compared to heuristic search hold for problem spaces: the set of operators is small and finite (or finitely generated); a new knowledge state is produced every few seconds or so; and backup (the set of elements that can be returned to) is very restricted. Also, as we will illustrate below, the Newell and Simon definition allows the existence of plans that can give varying amounts of direction to the search. The instantiation of the problem space concept for Studnt presented below has ordered components (3) and (4) before (1) and (2); it has combined (1) and (5), since there is in Studnt no need for any distinction in knowledge states; and it has added two components (e' and f') whose presence will be further discussed below. The following gives Studnt's problem space by referring to the model of the Ns above.

a'. The initial state of knowledge is statement a.

b'. The problem or desired state is b.

c'. Elements, or knowledge states: the partially processed input string + all of the internal symbol structures pertaining to the problem.

d'. The operators, which produce new elements:

  i. initial-scan operator set: transformations, dictionary tags, and segmenting: statements c, d, f, g, n, and o.

  ii. FV-segmentation operator: m.

  iii. parsing operator: scanning and splitting chunks, building expressions: f, h-k.

  iv. variable-matching operator: l.

e'. Plans: e; sequencing implicit in g-m.

f'. Monitoring transitions to new knowledge states: p.

Two features of this problem space description deserve closer attention. First, something needs to be said to justify the size of the operators chosen, since the operators are sets of Ps. Studnt fortunately has a set of Ps that monitor the knowledge state as major new information comes in, the I Ps, providing a natural dynamic boundary for the operators. To briefly reiterate the function of the I's, they are connected with measuring the size of the space of possible outputs at any point in the process of solution. For instance, at some point, we may know that there are two equations, five operators, and four distinct variables, which determines a finite number of possible outputs (the task of Studnt being to reduce that number to one). Further support for this division into operators comes by assuming 50 milliseconds for each Working Memory action of the process, and then computing the time this gives for each dynamic operator segment. The result (measured on a typical example) puts the time within the three- to five-second range observed by Newell and Simon (1972) for comparable problem space operators in general human problem solving. In particular, on the problem TEST2, the P I3 fires about 30 times, and there are about 2100 Working Memory actions, giving 70 actions between firings of I3; 70 X 50 milliseconds = 3.5 seconds. These figures are approximate, and actually only about two-thirds of I3's firings are meaningful as operator boundaries (it fires more than once at some boundaries), but this still gives five seconds as the result.

The second feature of the problem space that needs to be discussed is the existence of plans, point e' (e. in the model). A plan is some kind of explicit control that guides the applications of operators (Newell and Simon, 1972, pp. 822-823). At one extreme of planning in this sense is a specific algorithm that is guaranteed to achieve the desired result. The main plan in Studnt is the initial scan, which rigidly controls the order of application of the operators by moving a scan pointer along the input string from left to right. A plan controls the ordering of the operators in the initial-scan operator set. If we remove the sequencing assumptions in these plans, we get a process with more of a heuristic search structure, with various orderings tried according to some search scheme, and with some way of ordering the resulting end products in order to pick the best. Some search is necessary as is illustrated by the phrase "30 per cent of". "Of" is changed to the operator "times" if preceded by a number, and "30 per cent" becomes ".30". Clearly two different results obtain depending on the order of testing for "per cent" and "of preceded by a number". An interesting problem for further research is the transition from a planless process to the final Studnt, and in particular, whether plans are added bit by bit, with processing taking advantage of pieces of plans wherever possible, and searching otherwise. To investigate this further, the PS formulation, with all control explicit in the data state and in P conditions, seems more suitable than standard control structures. Formulating Studnt as a problem space in this way serves to organize the model (at least, for purposes of exposition), it points out interesting research questions, and it makes contact with other research in problem solving that will be discussed in Section C.7.

## C.2. Knowledge interactions in forming a production: S13

We now present an example of the knowledge encoding process as it is envisioned for an important Studnt P. The implied form of the encoding process, however, is not nearly as important at this stage as the KSs themselves and how they can be seen to interact. The following briefly introduces the process, postponing a more exact discussion until examples are presented.

The knowledge in a P is built up around a particular KS, its <u>principal KS</u>. The P results as parts of the principal KS refer to subjects of other KSs, thereby causing them to <u>interact</u> with it, defining its terms and elaborating the conditions under which it applies. A particular N interacts with other Ns to give the total intention of the P. Qs and Zs are then added as required to define terms, to provide specific techniques, and to make contact with the control structure of the underlying language. This process will now be illustrated by examining S13 in detail. In case the reader loses the overall structure of the following details, the material is summarized in Figure C.1 at the end of this subsection, and Section C.3 gives a summary in a different form.

S13 is a P that controls the initial scan of the input problem, invoking the transformation process and doing some bookkeeping on the string elements scanned.

```
S13; "TF SCAN" :: TGSCANFIN2(X) & LEFTOF(X,Y) & NOT ISDELIM(X) & ISSCANCHUNK(C)
               & CHUNKLEN(L)
          => TFSCAN(Y) & TFSCANFIN(Y) & INCHUNK(X,C) & CHUNKLEN(L+1)
               & NEGATE(1,5) & NOT TGSCAN(X);

          where NEGATE(1,5) ≡ NOT TGSCANFIN2(X) & NOT CHUNKLEN(L)
```

The principal KS for S13 is NS11[⊛]:

NS11 THE INITIAL SCAN PROCEEDS FROM LEFT TO RIGHT IN THE PROBLEM STRING, PERFORMING THE FOUR FUNCTIONS[⊛⊛] AT EACH POINT IN TURN, AND ADDING EACH WORD SCANNED TO THE CURRENT CHUNK.

The first phrase brings in Q4:

Q4 THE PROCESS OF SCANNING INVOLVES MOVING A SCAN POSITION FROM AN OLD POSITION TO A NEW ONE.

To determine the old position, use is made of TGSCANFIN2:

TGSCANFIN2(p) completion of initial scan processing at p, ready to move scan pointer.

The new position is determined by using Q8 which brings in[⊛⊛⊛] LEFTOF:

Q8 PROBLEM STRINGS AND SUBSTRINGS ARE SEQUENCES OF WORDS, READ FROM LEFT TO RIGHT, WITH EACH WORD DIRECTLY TO THE LEFT OF THE WORD FOLLOWING IT.

LEFTOF(p1,p2) p1 is directly to the left of p2.

This has determined everything relevant to the old position of the scan pointer,

[⊛] Ns are given labels of the form N + initial of a P group + number + occasionally a letter.

[⊛⊛] These are defined by separate KSs presented below.

[⊛⊛⊛] Some of the connections between KSs and between KSs and predicates may require free interpretation and detective work on the part of the reader. It is beyond the present scope and purpose to be more precise.

represented by the first two LHS conjuncts in S13. At a knowledge level that is suppressed here, it is understood that "old" would imply something in the condition (LHS), whereas the "new" refers to something in the action side of the P. What actually goes into the action side for the new pointer position depends on parts of NS11 that will be taken up later, after the interactions from what has been done so far have been discussed.

Now, the initial scan does not always proceed unconditionally, as stated by NS12:

NS12 WHEN THE END OF A CHUNK IS SCANNED, THE CHUNK IS COMPLETE, AND THE INITIAL SCAN IS INTERRUPTED FOR THE CHUNK SPLITTING PROCESS.

This interaction results, by indirection, in the third LHS conjunct. First there is an association to NS7, which defines how the end of a chunk is recognized:

NS7 WHEN A PERIOD WITH A DELIMITER TAG IS SCANNED, THE END OF THE CURRENT CHUNK HAS BEEN REACHED, IF THE CHUNK IS NOT AN FV CHUNK.

Using the meaning of ISDELIM, we get the third conjunct:

ISDELIM(p) p is a delimiter.

Here, a choice was made on whether the ISDELIM argument should be X or Y, that is, whether to interrupt the scan before or after looking at the delimiter of the chunk. The choice of X, namely the element just passed, follows from consideration of Q14 (which the knowledge encoding process would consult every time such a condition were tested):

Q14 DURING A SCAN PROCESS, WHEN A CONDITION IS STATED IN TERMS OF THE POSSIBLE OUTPUT OF SOME PROCESS THAT IS APPLIED AT EACH SCAN POINT, THE TEST FOR THAT CONDITION AT A PARTICULAR POINT SHOULD⊗ BE DEFERRED UNTIL THE SCAN HAS PASSED THE POINT.

In this case, one example of a relevant Studnt transformation is stated by NT25:

NT25 ", AND" TRANSFORMS TO "PERIOD".

We now proceed to the second phrase of NS11, which refers to performing four functions in turn. This is elaborated by Q5:

Q5 APPLYING A NUMBER OF FUNCTIONS IN TURN MEANS TO APPLY THE FIRST, AND WHEN THAT IS DONE, APPLY THE SECOND, AND SO ON.

So we need to know what the first function is:

NS1 THE FIRST FUNCTION OF THE INITIAL SCAN IS TO APPLY TRANSFORMATIONS AT EACH POINT IN THE SCAN.

Since we're doing a sequence of functions, we look at:

⊗ This kind of imperative language is typical of expressing KSs as if to an encoding process.

Q15 WHEN A SEQUENCE OF ACTIONS IS TO BE PERFORMED, MORE FLEXIBILITY[⊛] IN ALTERING THE COURSE OF THAT SEQUENCE OBTAINS BY BREAKING IT INTO SEPARATE STEPS, EACH REQUIRING AN INITIATE SIGNAL AND HAVING A COMPLETION SIGNAL; THIS BREAKING INTO STEPS IS ESPECIALLY USEFUL FOR LONGER SEQUENCES WHERE UNDER VARIOUS CONDITIONS, DIFFERENT ELEMENTS OF THE SEQUENCE ARE ACTUALLY EXECUTED.

This gets us to the use of TFSCAN and TFSCANFIN:

| | |
|---|---|
| TFSCAN(p) | signal to initiate check for string transformations at p. |
| TFSCANFIN(p) | signal completion of TFSCAN at p, ready for next step in the scan process. |

We use two signals because of:

Q24 WHEN THERE ARE MANY MORE WAYS OF COMPLETING A PROCESS EVOKED BY AN INITIATE SIGNAL THAN WAYS OF INITIATING IT, THE COMPLETION SIGNAL SHOULD BE EMITTED AT THE SAME TIME AS THE INITIATE SIGNAL, IN SUCH A WAY THAT THE INITIATE SIGNAL IS EXAMINED FIRST.

Since the order of consideration of these two insertions is critical, we must make use of:

Z2 THE FIRST TWO RIGHT-HAND-SIDE INSERTIONS ARE ORDERED AT THE TOP OF :SMPX; WHEN IT IS DESIRED TO DO ONE THING FOLLOWED BY ANOTHER, ORDER THE "INITIATE" SIGNALS ACCORDINGLY.

So, now we have the first two conjuncts of the RHS.

The final phrase of NS11 deals with noting that each word scanned is part of the current chunk. This cannot be unconditional, because of an interaction with NS10:

NS10 THE PERIOD AT THE END OF A CHUNK IS NOT INCLUDED AS PART OF THAT CHUNK OR ANY OTHER CHUNK.

This associates first to NS7 (see above), which says we're testing on "period". By the same reasoning as used before, this exclusion also has to be done after the scan on a position is done, so the NOT ISDELIM test serves a double purpose. To add to the current chunk, we need to know what it is:

ISSCANCHUNK(c) c is currently being scanned; it is not an FV.

This is the fourth LHS conjunct, and the act of noting is taken care of by the third RHS conjunct, which uses:

INCHUNK(p,c) element at p is in c.

[⊛] The Qs at times express qualitative goals like flexibility and efficiency, rather than simply giving absolute direction.

The use of ISSCANCHUNK allows us to clean up a loose end regarding the use of NS7. We must verify that in fact the end of the chunk has not been reached, and the NOT ISDELIM will work, provided this isn't an FV chunk; the definition of ISSCANCHUNK guarantees it.

This takes care of the central action with respect to NS11. It remains to consider some other associations which are related but are less essential to the main process. NI7 has to do with scanning, in fact, with the number of words scanned:

NI7 THE LENGTH OF THE PART OF THE PROBLEM AS YET UNSCANNED CHANGES EACH TIME A NEW OPERATOR, EQUATION, OR PERIOD IS SCANNED, AND IT CHANGES BY THE NUMBER OF WORDS SCANNED SINCE THE LAST CHANGE OR SINCE THE BEGINNING OF THE PROBLEM.

CHUNKLEN is the counter:

CHUNKLEN(n) current length of the current scan chunk is n.

To change a counter, we need the old value in the LHS, with the new value as part of the RHS. Q6 requires us to delete the old value of the counter:

Q6 WHEN A VALUE OF A COUNTER IS CHANGED, THE OLD VALUE SHOULD BE REMOVED.

This gets the sixth RHS conjunct.

We have not mentioned the fifth and seventh RHS conjuncts, whose purpose is to erase old scan signals. The appropriate KS:

Q3 FOR STORAGE EFFICIENCY, PROGRAM SEGMENTS THAT RESPOND TO SCAN SIGNALS OF THE "COMPLETION" TYPE SHOULD ALSO REMOVE THE CORRESPONDING "INITIATE" TYPE, AS WELL AS REMOVING THE USED "COMPLETION" SIGNAL, IF IT IS POSSIBLE THAT NO PROGRAM SEGMENT RESPONDS TO THE INITIATE SIGNAL.

There are other KSs that deal with the initial scan, which would be examined, but rejected, in the process of building S13.

NS2 THE SECOND FUNCTION OF THE INITIAL SCAN IS TO APPLY AGE-PROBLEM TRANSFORMATIONS, IF THE PROBLEM IS AN AGE PROBLEM, AT EACH SCAN POINT.

NS3 THE THIRD FUNCTION OF THE INITIAL SCAN IS TO PUT DICTIONARY TAGS ON WORDS AS EACH WORD IS SCANNED.

NS4 THE FOURTH FUNCTION OF THE INITIAL SCAN IS TO CHECK FOR A NEW HIGH PRECEDENCE WITHIN THE CHUNK BEING SCANNED, IF THAT CHUNK IS NOT AN FV CHUNK AS EACH WORD IS SCANNED.

NS5 THE FOURTH FUNCTION OF THE INITIAL SCAN IS TO APPLY THE FV TRANSFORMATIONS, IF THE CHUNK BEING SCANNED IS AN FV CHUNK, AS EACH WORD IS SCANNED; AN FV TRANSFORMATION IS ANY OPERATION THAT DEALS WITH THE DETERMINATION OF FV CHUNKS.

NS6 A CHUNK THAT STARTS WITH A WORD THAT IS A QWORD IS AN FV CHUNK.

NS8. THE FIRST CHUNK TO BE SCANNED STARTS IMMEDIATELY TO THE RIGHT OF THE LEFT END OF THE PROBLEM STRING.

NS9 WHEN THE END OF ONE CHUNK IS REACHED, ANOTHER BEGINS IMMEDIATELY, UNLESS THE RIGHT END OF THE PROBLEM STRING HAS BEEN REACHED.
NS13 THE LAST CHUNK IN A PROBLEM IS ALWAYS AN FV CHUNK.

NS2 through NS5 are rejected because they deal with functions of the scan other than the first. NS6 and NS13 are rejected because the QWORD tag is the result of the third scan function, and is thus unavailable. NS8 is relevant, and interacts with NS11 to produce another P, S10. NS9, NS10, and NS12 (the last two were displayed previously) do not add to the action because of the exclusion of their conditions with the third LHS conjunct.

Figure C.1 summarizes the interactions between the KSs that form S13 as described above. Each arrow represents an interaction, with its origin at the KS (or predicate, in one case) that initiates the interaction by requiring further elaboration.

## C.3. Summaries of interactions for selected productions

This subsection gives summaries of the formation process for a representative set of Ps. Since each summary lists only a P and its principal KS, the reader must refer to Appendix F, which lists the KSs in full, in order to follow the detail.

Each summary starts out with a listing of the P and its principal KS. If the P has any macros, their expanded form is given. The body of the summary is organized into "sentences", delimited by ".", broken into segments delimited by ";". A sentence represents closely interrelated processing, with each segment dealing with the determination of a set of conjuncts of the P. The conjuncts are referred to by labels such as "L1" and "R3", which stand, respectively, for "first LHS conjunct" and "third RHS conjunct". In counting in RHSs, EXISTS conjuncts are ignored. Lines giving macro expansions also give labels for the conjuncts in []'s to aid in determining referents of labels for the conjunctions containing the macros. Within segments, "&" is used to indicate "interacts or combines with", a binary operator on KSs; "->" is used for "associates to". "&" has a higher binding power than "->", i.e., a & b -> c & d is really (a & b) -> (c & d). These are, of course, to be interpreted loosely. Each sentence has as subject its first element; segments that start with "&" or "->" implicitly have an occurrence of the subject.

The summary of S13 appears first, so that the reader may become accustomed to the notation on familiar material. The meaning of "excitatory interaction" is explained below.

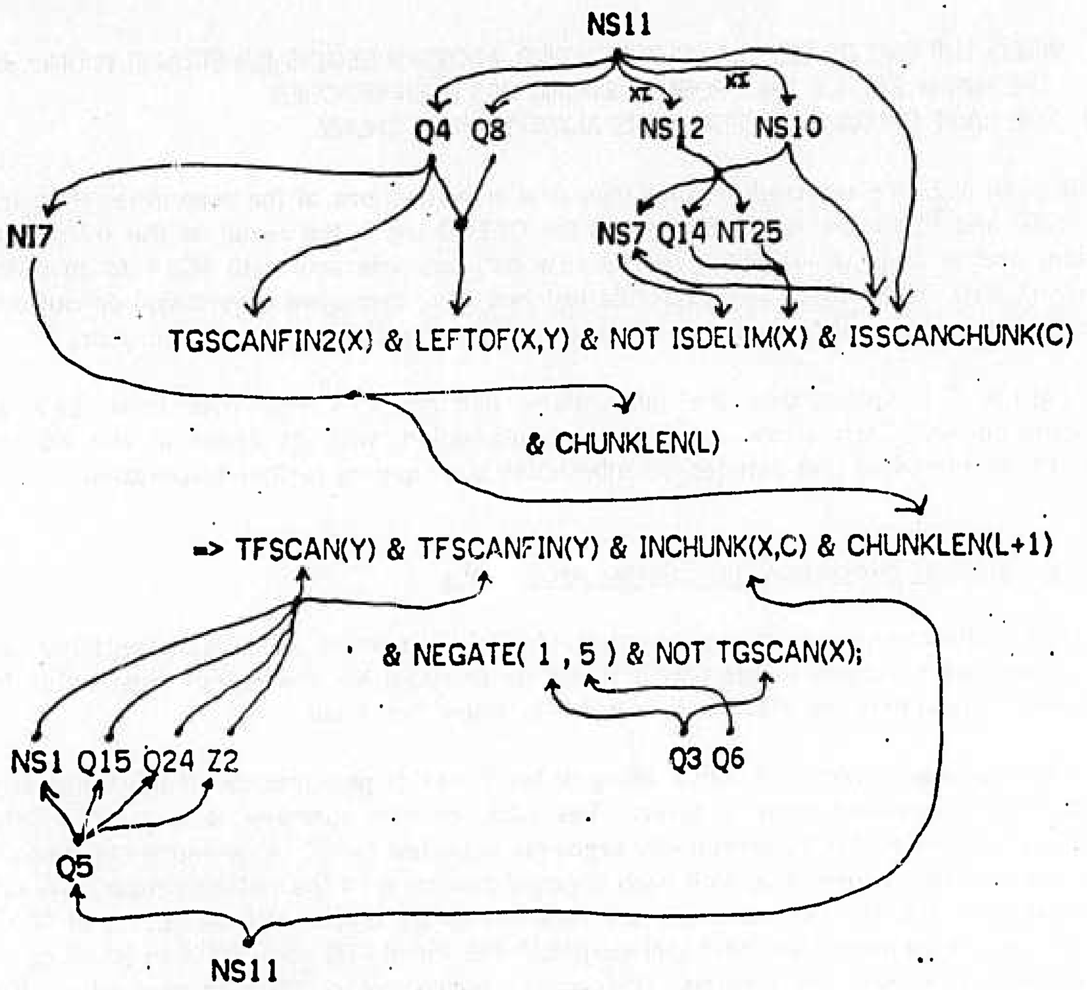

NS11
XI
XI
Q4 Q8
NS12
NS10
NI7
NS7 Q14 NT25
TGSCANFIN2(X) & LEFTOF(X,Y) & NOT ISDELIM(X) & ISSCANCHUNK(C)
& CHUNKLEN(L)
=> TFSCAN(Y) & TFSCANFIN(Y) & INCHUNK(X,C) & CHUNKLEN(L+1)
& NEGATE( 1 , 5 ) & NOT TGSCAN(X);
NS1 Q15 Q24 Z2
Q3 Q6
Q5
NS11

Figure C.1 Knowledge interactions in forming S13

Summary for S13:

```
S13; "TF SCAN" :: TGSCANFIN2(X) & LEFTOF(X,Y) & NOT ISDELIM(X) & ISSCANCHUNK(C)
                & CHUNKLEN(L)
        => TFSCAN(Y) & TFSCANFIN(Y) & INCHUNK(X,C) & CHUNKLEN(L+1)
                & NEGATE(1,5) & NOT TGSCAN(X);

        where NEGATE(1,5) ≡ NOT TGSCANFIN2(X) & NOT CHUNKLEN(L) [R5, R6]

principal (model statement e.):
NS11  THE INITIAL SCAN PROCEEDS FROM LEFT TO RIGHT IN THE PROBLEM STRING,
      PERFORMING THE FOUR FUNCTIONS AT EACH POINT IN TURN, AND ADDING EACH
      WORD SCANNED TO THE CURRENT CHUNK.

        first phrase: Q4 -> L1; Q4 & Q8 -> L2;
                excitatory interaction: NS12 -> NS7 -> L3;
                        Q14 & NT25 (& others) -> arg of L3.
        second phrase: Q5 -> NS1 & Q15 & Q24 & Z2 -> R1, R2.
        third phrase: L4, R3;
                excitatory interaction: NS10 -> NS7 & L4 def'n -> L3
                        (again, arg as above).
        Q4 -> NI7 -> L5, R4; Q6 -> R6.
        Q3 -> R5, R7.
```

The following summaries are given to indicate the uniformity and general applicability of the above knowledge encoding process to all of Studnt's Ps. T12 is a typical initial-scan transformation P, with much simpler structure than S13. M10, C60, and C75 deal with the process of breaking down chunks into operators and operands, and then putting the completed expressions together to form an equation. F60, F70, and F75 illustrate the processing of one type of FV form. These examples illustrate the application of over half of the Qs, and introduce twenty new Ns.

The examples also include three "bugs" which were discovered by the knowledge analysis (see C75, F70, F75). These are bugs from the standpoint of the analysis, not defects in the actual output of the program. The first involves having two Ps with overlapping conditions, where a combination of the two into one is more appropriate, and is dictated by the analysis. The second bug is an inconsequential incorrect ordering of RHS assertions. The third seems more serious, since it is an omission of updating the element that denotes which chunk is the current scan chunk. However, its bad effects are cancelled by the failure of other Ps to check for or make use of that information. A more general discussion of the types of bugs encountered in the process of doing the knowledge analysis is below, Section C.4.

Summary for T12:

```
T12; "TWICE->TWO TIMES" :: TFSCAN(V?-1) & STRINGEQ('(TWICE),X,Y)
      => MODLEN(1) & EQ2(V?-1) & WORDEQ(V?-1,'2)
            & NOT WORDEQ(V?-1,'TWICE) & STRINGINS('(TIMES),V?-1,Y)
            & NEGATE(ALL,-2);

      where  STRINGEQ('(TWICE),X,Y) ≡ LEFTOF(X,V?-1) & EQTWICE(V?-1)
                 & LEFTOF(V?-1,Y)        [L2, L3, L4]
            STRINGINS('(TIMES),V?-1,Y) ≡ EXISTS(T1) & LEFTOF(T1,V?-1)
                       & EQTIMES(V?-1) & WORDEQ(V?-1,'TIMES)
                       & LEFTOF(V?-1,Y)        [R5-R8]
            NEGATE(ALL,-2) ≡ NOT TFSCAN(V?-1) & NOT EQTWICE(V?-1)
                       & NOT LEFTOF(V?-1,Y)    [R9, R10, R11]
```

principal (model statement c.):
NT12 "TWICE" TRANSFORMS TO "2 TIMES".

```
NT12 -> L3, R2, R3, R6, R7; (checks other NT's, by Q11, but no effect);
      Q8 -> L2, L4, R5, R8; Q12 -> R4, R10, R11.
"transforms to" -> NS1 -> L1; Q7 -> R9; Q9 -> args of R2, R3, R4;
      NI9 -> R1 (order determined by NI10 & Z1).
```

Summary for M10:

```
M10; "CONN =" :: EQIS(X) & HASIS(C,X) & LEFTOF(X,A2)
           & NOT EQMULTIPLIED(A2) & NOT EQDIVIDED(A2) & NOT EQINCREASED(A2)
     => NEWEQN(X) & CSPLIT(C,X,X) & HASOP(C,'EQUAL) & NEGATE(2);

     where NEGATE(2) ≡ NOT HASIS(C,X)             [R4]
```

principal (model statement i.):
NC4 A CHUNK WITH A HIGHEST-PRECEDENCE OPERATOR MARKED, EXCEPT "SQUARE" AND "SQUARED", IS SPLIT INTO TWO NEW CHUNKS, WITH THE LEFT END OF THE LEFT CHUNK THE SAME AS THE ORIGINAL, RIGHT END OF THE LEFT CHUNK THE WORD DIRECTLY TO THE LEFT OF THE PHRASE REPRESENTING THE OPERATOR, LEFT END OF THE RIGHT CHUNK DIRECTLY TO THE RIGHT OF THE PHRASE REPRESENTING THE OPERATOR, AND RIGHT END OF THE RIGHT CHUNK AT THE RIGHT END OF THE ORIGINAL CHUNK.

```
NC4 -> L1, L2; & NM1 & NC5 -> R3; & Q16 -> R2.
string in condition -> Q11 -> inter with NC1 -> L4, L5, L6;
      & Q8 -> L3.
"equal" in NM1 -> NC12 -> NI1 -> R1 (order by NI10 & Z1).
"split" in NC4 -> Q13 -> R4.
```

Summary for C60:

```
C60; "SPLIT CHUNK" :: CSPLIT(C,LOCL,LOCR) & LEFTOF(X1,LOCL) & LEFTOF(LOCR,X2)
            & LABELU(C,N,P) & MXCPRIOR(M)
      => EXISTS(CL,CR) & NEWPLOP(C) & RRENAME(X2,C,CR) & LRENAME(X1,C,CL)
            & LABELU(CL,N+1,C) & LABELU(CR,N+1,C) & HASCPRIOR(CL,M+2)
            & HASCPRIOR(CR,M+1) & MXCPRIOR(M+2) & CHUNKENDL(X2,CR)
            & CHUNKENDR(X1,CL) & NEGATE(1,2,3,5);

      where NEGATE(1,2,3,5) ≡ NOT CSPLIT(C,LOCL,LOCR) & NOT LEFTOF(X1,LOCL)
                   & NOT LEFTOF(LOCR,X2) & NOT MXCPRIOR(M) [R11-R14]
```

principal: NC4 (see above)
    NC4 -> NC5 & Q16 & Q8 -> L1, L2, L3.
    "new chunks" -> NR1 & NR2 & Q53 -> NC15 -> Q19 -> R2, R3.
    NC5 -> Q20 -> L4, R4, R5, L5, R6, R7, R8.
    renaming -> Q21 -> R9, R10.
    operator placed in expression -> NI1 -> NI10 & Z3 -> R1,
        order of R1, R2, R3.
    Q18 -> R11. Q17 -> R12, R13. Q6 -> R14.

Summary for C75:

```
C75; "FINISH SEG =" :: LABELU(C,N,P) & LABELF(C1,M,C) & LABELF(C2,M,C)
            & HASOP(C,X) & SATISFIES(X,X EQ 'EQUAL) & HASCPRIOR(C1,PR1)
            & HASCPRIOR(C2,PR2) & SATISFIES2(PR1,PR2,PR1 ?*GREAT PR2)
            & SATISFIES(M,M EQ 2) & HASEXPR(C1,Y) & HASEXPR(C2,Z)
      => NEWREFEXPR(C1) & HASEXPR(C,<X,Y,Z>) & LABELF(C,N,P) & NEGATE(1);

      where NEGATE(1) ≡ NOT LABELU(C,N,P)    [R4]
            and <X,Y,Z> converts to the LISP expression (LIST X Y Z)
```

principal (model statement k.):
NC11 AN EXPRESSION IS A TREE STRUCTURE OF THE FORM (a b c) WHERE a IS THE OPERATOR, b IS THE TREE EXPRESSION FOR THE LEFT OPERAND, AND c IS THE SAME FOR THE RIGHT OPERAND.

    NC11 -> L4, L10, L11, R2.
    "tree structure" -> Q20 -> NC5 & NC17 -> L1, L2, L3, L6, L7, L8,
                L9, R3; Q33 -> R4.
    "left operand" -> NC14 -> L5, R1. (conditional, others are C70, C78.)
    (in the given KS framework, NC12 should also be included; reason
        for its absence is related to the growth of the program:
        C70 - C78 were not split into the three conditions originally,
        so that C90 was necessary.)

Summary for F60:

```
F60; "FIND FV" :: EQFIND(X) & FVSCAN(X) & ISSCANFV(C) & CHUNKENDL(X,C)
        & LEFTOF(X,Y)
     => CHUNKENDL(Y,C) & RTANDPERGOING(C) & NEGATE(2,4);

     where NEGATE(2,4) ≡ NOT FVSCAN(X) & NOT CHUNKENDL(X,C)   [R3, R4]
```

principal (model statement m.):

NF8 A SENTENCE WHICH STARTS WITH "FIND" HAS FV CHUNKS STARTING AFTER THE "FIND" AND SEPARATED BY "AND" , AND IT ENDS WITH "PERIOD".

```
     NF8 -> L1, L4.
     "FV" -> NS5 -> L2, L3; transformation -> Q7 -> R3.
     "find" adjacent to start -> Q8 -> L5.
     removal of "find" -> Q23 -> L4, R1; last phrase of Q23 -> R4;
          & Q10, inhibited by NF9.
     separator after chunk -> later in scan -> Q22 -> R2.
```

Summary for F70:

```
F70; "&-. ." :: FVSCAN(X) & RTANDPERGOING(C) & EQPERIOD(X) & LEFTOF(W,X)
     => ISVARCHUNK(C) & CHUNKENDR(W,C) & FVSCANEND(X,C) & NEGATE(1,2);

     where NEGATE(1,2) ≡ NOT FVSCAN(X) & NOT RTANDPERGOING(C)  [R4, R5]
```

principal: NF8, see above.

```
     NF8 & Q22 -> L2, L3. "FV" -> NS5 -> L1.
     "period" -> NF2 & Q23 & Q8 -> L4, R2;
          NS10 & NS11 & Q14 inhibits NOT INCHUNK(X).
     end of FV -> NF3 -> R1; -> Q16 -> R3; -> Q18 -> R5;
          -> NS12 -> Z2 -> order of R1, R3 (bug: R2 should be after R3).
     transform -> Q7 -> R4.
```

```
Summary for F75:

F75; "&-, &" :: FVSCAN(X) & RTANDPERGOING(OC) & EQAND(X)
                    & LEFTOF(W,X) & LEFTOF(X,Y)
          => ISVARCHUNK(OC) & CHUNKENDR(W,OC) & EXISTS(C) & NEWFV(C)
                    & ISFV(C) & RTANDPERGOING(C) & CHUNKENDL(Y,C) & NEGATE(1,2);

          where NEGATE(1,2) ≡ NOT FVSCAN(X) & NOT RTANDPERGOING(OC)  [R7, R8]

principal: NF8, see above.
          NF8 & Q22 -> L2, L3; & Q25 & Q8 & Q23 -> L4, L5, R2. "FV" -> NS5 -> L1.
          separator -> NF3 -> R1; -> Q25 & Q1 & Q16 -> R4, R6; -> Q22 -> R5;
                    Q25 new chunk -> NI1 -> R3.
                    (bug: missing ISSCANFV update, apparently a serious bug,
                              but it works ok because other Ps don't check)
          transform -> Q7 -> R7; R5 & Q34 -> R8;
                    end of chunk -> NS12 -> R1 before R4.
          NI1 -> NI10 -> Z3 -> order of RHS, except bug, should be R3. R1, R4, R2.
```

## C.4. <u>Summary comments on the details of the analysis</u>

This subsection discusses in a more general way the knowledge encoding process revealed in the examples just given. Then, there is a short discussion of the bugs that were detected in carrying out the analysis for all of Studnt. The reader will need to refer to Appendix F to follow the examples used as supporting evidence.

The knowledge encoding process starts out with statements that are close to the abstract model characterization of the target process. That is, particular KSs are selected to be principal KSs on the basis of their plan-like nature, as opposed to being simple assertions of facts. For example, among the NS's, NS1-5, 8, 11, 12 and 14 are used as principal KSs, while NS6, 7, 9, 10 and 13 are not (actually the inclusion here of NS6 and NS9 needs to be qualified, see below). Similarly, NC2, 4, 6, 7, 10 and 11 are the NC's that are principal. It is evident from these examples, however, that it may be impossible in general to decide which KSs can be principal without fully working out the interactions, to see how the KSs stand in relation to each other. Note that model statement g. (Section C.1) is elaborated almost entirely by non-principal KSs. This may indicate that the structure of the model can be helpful in distinguishing principal from non-principal. Another common feature of non-principals is the use of phrases like "whenever": NM12, NM13, and NF9 are examples.

Once a principal KS has been chosen, interactions of three main sorts occur: definitional, excitatory and inhibitory. A <u>definitional interaction</u> is an interaction in which one KS defines a term in another. We have seen a definitional interaction in the use of Q5, dealing with sequential application of functions, which is further elaborated definitionally using NS1, ultimately obtaining conjuncts R1 and R2 of S13. An <u>excitatory interaction</u> is an interaction between KSs that results in additional specific conditions for the application of the principal KS, e.g., NS12 interacts with NS11 to result in conjunct L3, a condition element that excludes the normal scan processing when a delimiter is seen. An inhibitory interaction, on the other hand, is one that suppresses elements of Ps; an illustration is the interaction of Q10 and NF9 in the summary for F60 above, which suppresses rearranging scan pointers on the removal of "find" from an FV chunk.

The Q KSs interact according to the definitional type of interaction, above, and perform two other types of function: erasing unneeded Working Memory items and adding programming techniques. These three broad types of Qs are discussed further below, but at present we consider how they come to be applied. Erasing Qs are applied after other interactions have been completed, and the application is fairly direct from their statement. For instance, Q6 applies in the S13 example to delete the old value of the counter when a new value is computed. The programming-technique Qs are more central to the process, as is illustrated by the episode which results in conjuncts R1 and R2 of S13. NS11 speaks of performing some actions in sequence (paraphrasing freely), so that Q15 is directly applicable, along with Z2, by virtue of stated application conditions. The justification of Q24 is not nearly so direct, involving aspects of the process which are more problematic. That is, it assumes knowledge of a non-local sort, namely that there are many transformations (NT's). It also is complicated by being cast in PS-like terms, so that perhaps it should be classed as a Z not Q. These issues will be discussed further below, and need not detract from more general considerations of how Qs and Zs come into the interaction process, as intended by the use of the S13 episode above. The Z KSs interact in ways similar to the programming-technique Qs.

The process of selecting principal KSs and carrying out interactions can be viewed as a variant of a goal-subgoal scheme, where a goal might be to form a P from some KS, with subgoals generated during the interactions and stacked for later consideration (cf. a similar organization, "contingency planning", in Buchanan's (1974) automatic programming system). These subgoals arise when interactions are discovered which require KSs to be considered as principal KSs, which might not have otherwise been considered as such. Ps that result can be termed subsidiary Ps. One example of a subsidiary P whose "principal" occurs elsewhere as a non-principal is S65, with principal NS9 (this is, in fact, one of the Ns listed previously as exemplary non-principals). Another class of subsidiary Ps responds to store-recompute decisions, whereby some aspect is computed by the subsidiary P and stored as a data element to avoid repeating the computation. For example, S60 is built around NS6, which is more assertive than plan-like and thus would not ordinarily be a principal KS. Certain kinds of programming techniques require coordination of more than one P. The primary example of this is looping, which requires a set of Ps representing the body of the loop and another set representing its termination. In this case a goal-subgoal organization could be used to keep track of the disjoint pieces of program.

Analyzing the Ps from the standpoint of the KSs in them has resulted in the discovery of bugs, of the following five varieties: (1) omission of updates to data structures that turned out to be redundant (for example the group, taken together, F75, S15, V25 ff); (2) failure to delete properly (C2, F50); (3) RHS ordering not correct, with some assertions not important to order placed before ones whose order is important (F70, F75); (4) separation of Ps, where combination is possible (C75, C90); (5) awkward combination of Ps, where separation would result in less complexity in P conditions (S17-S35 could be re-organized). The first type, although occurring only once, seems to be the most serious (it was discussed in Section C.3). The primary reason that the particular example didn't result in errors by the program is the redundancy of the Working Memory, that is, the Ps that processed the partially erroneous data did not check it for complete consistency. The redundancy is due to the overly cautious nature of the problem space plans, which dictated the structures to be built during initial scan (NS11), and the lack of the checks on the data structures is due to insufficient tendency of the Ns (in this case, NV6) to be associated with checks on data consistency.

The basic issue here seems to be that in analyzing how a program ought to be written within the present scheme, and in comparing that with the actual program, the actual program falls short of expectations in ways that can not be tolerated in the output of some automatic programming procedure. That is, an automated procedure to produce programs in the present scheme would need to (and could be expected to) exercise more caution in such situations, producing programs as close to being correct as is possible. A further consideration is that the result of the bug's presence is that not everything is explicit. That is, effects of changes to the program would not have been noticed at locations where no checks occurred but things were by default assumed in good shape. In general, this is a bad practice, since PSs are capable of the desired explicitness, and should exploit it. It is clear that the second, third and fourth types of bugs are similar to the first in these respects. The fifth class of bug is really a matter of programming techniques that might have been used to result in less complex conditions, and in general, fewer Ps, since separating conditions into distinct sets of Ps makes the possibilities additive instead of multiplicative. In some places in the program this principle was applied, but the application was not uniform. The knowledge encoding process is expected to involve some search in investigating interactions of KSs, in order to decide between alternative expressions of program segments.

## C.5. Further characterizations of the knowledge statements

We now return to the topic of the partition of KSs into the Ns, Qs, and Zs, which was introduced at the beginning of this section. The coherence of the N class has already been demonstrated by presenting a model for the Ns and by associating that model with a problem space formulation of Studnt's problem solving. The Qs and Zs do not appear to be coherent enough to construct a model at this time; the structure of the Q and Z knowledge will only emerge after a fuller set of such statements has been determined. This subsection will group the Qs and Zs into some broad categories, and then discuss the N-Q-Z partition with regard to substitutibility of other such sets of KSs for the present ones, modularity of knowledge, and augmentation of Studnt and how it affects the various classes of KSs.

The Q KSs can be divided into 3 broad types: definitional [Q4, 5, 8, 25, 53], erasing [Q3, 6, 7, 12, 13, 17, 18, (23), 31, 33, 34, 37, (38), 39, 41, 46, (51)] and programming techniques [all the rest]. Some of them have secondary meanings which belong in a class other than the primary one, and this is indicated in the preceding and following lists by enclosing in parentheses. We have seen above that some of the Ns are also of a definitional type, so that we must distinguish between the two as follows. Definitions that are problem space dependent, e.g. "The end of a chunk is the delimiter, period", are classified as Ns. Qs are intended to be just the opposite, since they define entities that can be encountered in many task environments, such as strings and scanning.

The Qs can also be characterized by primary topic, as follows:

a. Sequencing, applying functions, communication between processes, use of signals [5, 15, 16, 18, 19, 24, 28, 37, (38), (39), (41), 42, (49), 50, (51)].
b. Scanning [3, 4, 14, 22, (26), (31), 44, 48].
c. Transformations on strings [7, 9, 10, (26), 31, 42, (52)].
d. Numeric: counting, ordering, and finding maxima [6, 26, 27, 29, 30, 32, 40].
e. Clean-up operations, attribute erasure [13, 33, 34, 38, 39, 41, 51].
f. Strings [8, 11, 12, (22), 43, 52].
g. Looping [(21), 36, 45, 46, 49].
h. Structures: tree, linear, splitting linear ones, separators, renaming [20, 21, 23, 25, 53].
i. Initialization [1, 2, (27), 47].
j. Use of a dummy as a place-holder [35].

Topic a. is the topic which may appear to have the most dependence on PSs, so that something more is required to justify any claims for generality. That topic's Qs are stated in terms of processes with two kinds of associated signals, initiation and completion, with the former emitted by the evoking process, and the latter by the evoked one. Signals are taken to be entities that can be processed, cancelled, and conditionally emitted. The crucial assumption is that signals can be emitted to be processed in a particular order, that is, that many can be emitted simultaneously, with processing of those in some specified order. This last assumption is the attribute that is most difficult to justify as appropriate to a non-production-system context. Further study will reveal if this is a major difficulty or not. Three of the KSs in particular are offensive in regard to possible scope limitations: Q15, Q24, and Q40. The first two use the signal order attribute just mentioned. Q40 specifically mentions "multiple firings", which is recognizable as referring to firings of Ps. But the statement is referring to a more general concept, that of synchronizing the results of asynchronous processes, so that the choice of words may be questionable, but the concept maintains the desirable degree of generality. One further point is that the erasure component of Qs is not at all necessary (at least, visibly) in languages which automatically discard local memory contexts, or which don't require explicit data signals for control primitives.

The Zs can be grouped into five topics:

a. Order in RHSs of Ps [1, 2, 3, 11].
b. Re-assertion of instances, use of :SMPX [4, 7, 8].
c. Peculiarities of the match, especially its being keyed to new data [5, 6].
d. Contradictory actions possible [9].
e. Specific control of looping [10].

The following model of Psnlst, although not fully general, suffices to explain the content of the Zs. Psnlst is a PS interpreter in which Ps detect conditions in an associative unstructured Working Memory. As a result of detecting conditions, specific actions are performed, consisting of additions to and deletions from the Working Memory. The Working Memory at any moment is partitioned into new data and old data, where new data are elements that have not been processed relative to specific Ps to which the elements may have relevance, i.e., Ps whose conditions may become true as a result of the elements. For a condition to be considered true, at least one element of it must match a new data element. The order in which new data elements are processed with respect to relevant Ps is determined by a stack, :SMPX, and the order of elements in the action sides (RHSs) of Ps determines order of placement in the stack. Elements which may have become old become new again by repeating their addition to the Working Memory (referred to as re-assertion). Each data element's first element is its predicate, and elements of the Working Memory are grouped by predicate. Predicates can be declared to be nonfluents, in which case data elements with those predicates never have the new status, i.e., no :SMPX entry is made for processing conditions relevant to nonfluents. Predicates are fluents, if they are not nonfluents.

Of the set of Zs three are related to the issue of whether there is some non-local knowledge in the Ps: Z5, Z6, and Z8. That is, these seem to require that one P knows what actions some others are performing, and perhaps how they're sequenced. This in fact is not the case, with one exception which can be avoided. Z8 is similar to Q11, in that it requires knowledge of other KSs, and need not be dependent on actual Ps. Z5 and Z6 are alike in that they can be handled in a very local manner, although one use of Z6 actually has a more global scope. That is, when a P wants to exclude firing again on data, part of which it has already processed, it can emit a signal specific to itself which indicates this, or it can include in its condition some part of its action which can be used for such an indicator. The use of Z6 (P V5) that violates localness (and which can be fixed in the former way) assumes that one signal it emits ultimately results in the change which is used in its condition to exclude spurious action later on.

Three aspects of the way the KSs have been partitioned indicate a wider applicability for the model and motivate the particular boundaries chosen. First, the division into Ns, Qs, and Zs is intended to be such that other analogous sets of KSs could be substituted with no interaction with statements in the other sets. For instance, we might want to use the Qs and Zs in conjunction with knowledge about solving logic puzzles, or we might want to program STUDENT in a different language. It turns out that this ideal is attained strongly in only one direction. For instance, changing to a different problem space would not affect the statements in the Q and Z sets, although the sets would probably need to be expanded with additional elements to meet different demands on technique. A change in the underlying programming language would not necessarily affect the Qs and Ns, although it is often the case that such changes come about in order to adapt fully to the available language facilities. In the case at hand we have two instances of this kind of language dependence. In the comparison above between STUDENT and Studnt, we saw how the change in language affected some of the plans in the problem space. We have also seen above how PS concepts may have weakly influenced how the Qs are stated. The clean substitutibility of sets of statements at the N level is really the most important and desirable form of substitutibility, since in a larger knowledge acquisition context, the other forms of change would never occur.

The second aspect of the N-Q-Z division is the issue of modularity of knowledge. A body of knowledge is modular if it has internal coherence or rich internal interconnectedness while relations to external knowledge are significantly fewer. Modularity is useful because it allows a body of diverse knowledge to be decomposed into units (modules) larger than primitive elements, making it more manageable and allowing structure to be made evident more easily. Individual KSs are hardly modular: they interact to a large extent with other KSs. But they do have a certain orderliness with respect to the containing knowledge structure as represented by models. So instead of individual KS modularity, we have model-level modularity, of two types. Within a model, there may be a partition that allows some relatively independent part to be taken as a unit and perhaps replaced as a unit. An example of this might be a major change to the way similarities of variables are determined (model statement I., Section C.1). The model as a whole might be taken as a unit and replaced. For instance, a shift to a different problem space might occur. The considerations raised above in connection with substitutibility apply to this case. This approach to modularity is speculative, and it depends on the exact form taken by models when the knowledge encoding and extracting processes become actual programs.

The third aspect of the way the KSs have been partitioned deals with augmentation of the set of Ns, rather than the larger operation of completely replacing it. One clearcut case of augmentation already exists in Studnt, namely the age-problem heuristics (A Ps). There are 19 Ns (all of the NA's plus NS2 and ND6) that are age-problem-specific, 11 such Qs (Q26, 31, 42-44, 47-52), and one Z (Z8). That is, those KSs were added to extend Studnt to the new set of tests (Test6, 9 and 10). The A Ps themselves use three Ns, 13 Qs and six Zs that are used elsewhere in Studnt, which indicates small N overlap but large Q and Z overlap. When we consider the age problems solved, we see that the A Ps were only about 8% of the total number of P firings, indicating a large overlap in processing with other problems. The conclusion from this is that augmenting the given framework to include a new class of problems can easily be seen as extending the knowledge sets involved, with a majority of new KSs in the N class. As long as the augmentation doesn't require major new kinds of processing (as sketched above, Section C.5), it can rely to a large degree on existing mechanisms. In fact, the original STUDENT design (and consequently Studnt's design) is such that the age problem augmentation was relatively easy to do, but this doesn't detract from the present conclusions, because the class of augmentations of the same type is large. Augmentations of a more difficult type (as defined in Section C.7) might have less Q and Z overlap.

## C.6. The knowledge extraction process

So far, our discussion has been oriented towards viewing Studnt as the result of a knowledge encoding process, but as stated in the introduction to this section, the knowledge was extracted from Studnt by an analysis. The primary attribute of the knowledge analysis is the many-many mapping between KSs and Ps, and to justify this we need to re-examine the knowledge extraction process.

Since the reader already has some familiarity with S13, we can use it as an example of how the form of KSs emerges from its content. We review what each conjunct contributes as follows:

L1: finished with initial scan at x, ready to move pointer.
L2: x is to the left of y.
L3: x is not a delimiter.
L4: current scan chunk is c.
L5: current length of scanned chunk is l.

R1: start transform check at y, the new scan pointer.
R2: finish transform check at y.
R3: x is in chunk c.
R4: current length of scanned chunk is now l+1.
R5: negate L1.
R6: negate L5.
R7: remove old scan-check signal for x.

From this description, we can sketch how the knowledge contained in S13 can be read off directly from the surface structure of the P. NS11 is composed of three phrases, two of which derive from L1 + L2 + R1, the third from L4 + R3. The first cluster says essentially that the scan is updated, left-to-right, and then the transform check is started. The second says that x becomes part of the current scan chunk. These elements fit together in such clusters by virtue of shared variables, x and y in the first case, c in the second, and by virtue of predicates with similar meanings. In the formation of NS11, Q4 and Q8 have been abstracted as separate definitions, since they are recognizable as potentially useful in many places. An exception to the scan process is given by L3, by virtue of its negative sign, so that it is known that some knowledge has interacted by specifying some incompatible action under the negated condition. From knowledge of the abstract model of the process, that negated condition is evidently an instance of the end of a chunk, so that NS12 is hinted at, using the definitional KS NS7. A further refinement of L3 is that its argument, x, carries some information, since without other considerations, y would appear to be equally possible (of course, an arbitrary choice might have resulted in x, but we must look first for some other justification). How that information is elaborated should be clear from the analysis of S13 that was carried out in detail above. Interestingly, the argument x of L3 provides a link to two actions, and the interaction with NS12 results only in the use of y in R1 which is linked to x by L2. It appears again in R3, so that another interaction is evident, this time having to do with adding elements to chunks, KS NS10. Another feature that can be read off from the P is the update of the length of the scanned chunk, with argument l linking L5 and R4. This link is expressed by NI7. Finally, the last three RHS assertions, R5-R7, are deletions, and lead to the formation of the appropriate Q KSs.

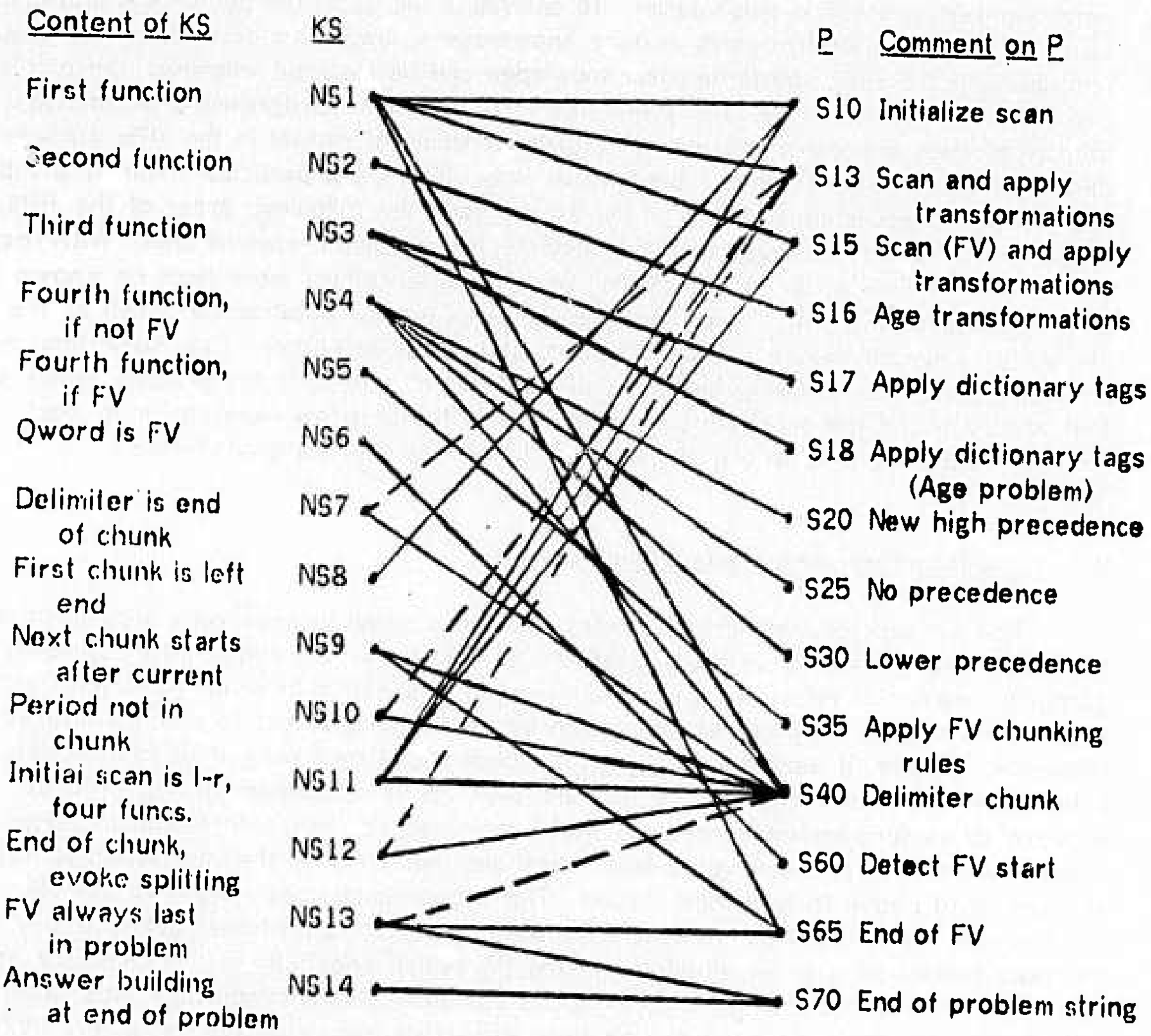
So, reading off what a P does gets a set of propositions, which are then taken singly as KSs, or, if several are so interdependent that they cannot stand alone, they are grouped as one KS. Support that some cluster is a meaningful grouping is gained from occurrences in many Ps, resulting in a certain economy of expression as the analysis is extended. The question of why the many-many mapping is obtained thus reduces to why the size of the P is what it is. S13 is the size it is because a certain number of things have to be done as the scan progresses, and they must be done before the process goes on. There is a good reason why it is less than elegant in operation if it is broken down into its component parts, with each a separate P. If each P did the thing stated by a single KS, the various Ps would be obliged to check each other's output, and at times to force retractions of certain actions. For instance, in S13, without explicit interactions with NS12, a signal would be emitted as if the scan were to continue, but that signal would be intercepted and delayed while the chunk splitting process were done. As things actually are, that condition is recognized before any signals are emitted, and behavior adjustment occurs appropriately. Breaking up a P into smaller ones would thus require extra KSs for the additional control. Clearly there is an optimum with respect to minimizing the number of KSs. Of course, matching overhead and efficiency would be affected by this change in organization, but that is a secondary concern at the moment. On the other hand, making Ps contain more KSs does not pay because one then has to multiply Ps in order to get all of the logical combinations of conditions. For instance, if three Ps perform one stage of a test, and four others perform another stage of the test, combining Ps might require as many as twelve Ps (where seven had sufficed) to handle all possible paths throught the two test stages. .

Figure C.2 illustrates the many-many mapping between Ns and Ps, for the S Ps, restricted to NS's. (NP's, NI's, Qs, and Zs are not shown; S20, S25, S30, and S40 use NP3, while S13, S15, S40, S60, and S65 use NI's).

Distributional data for the KSs over Ps supports the size that was chosen as a unit KS. This data is derived mostly from Appendix F, which gives the Ps that use each KS, and which has at its end a table that gives distribution frequencies for Ps having specific numbers of Ns, Qs, and Zs. The rest of the data comes from an inversion (not included) of that appendix, which gives the KSs associated with each P.

For Ns, nearly a majority (59 out of 154) are used in only one P, somewhat fewer are used in two (33), and fewer still in three or four (14 and 3, respectively). Ns that are used in more than four Ps are less numerous, with frequencies at or near zero. There are extremes, however: NI10 is used in 70 Ps (the maximum), and some others that are heavily used are NI1, NS1, NS3, ND13, and NI9. For Qs and Zs the distribution in frequencies is about the same (10) for uses in each category for 1 to 3 Ps, down to around 3 for 4 to 9 uses, and then at or near 0, with the maximum number of uses 105 for Q8 (other heavily used KSs: Z2, Q12, Z1, Q7, and Q18). Thus the distribution of Q and Z uses is somewhat flatter and more spread out than for the Ns, which is in accord with their being more generally applicable than the Ns. The high frequencies for low numbers of uses supports a unitary property for KSs, as opposed to compositeness. The many-many mapping of KSs to Ps is supported as follows. There are about 55 Ps for each frequency class for 1 to 4 KSs in each of the N and Q classes (accounting for a total of about 220 Ps). This means, for instance, that about 55 Ps have 2 Ns and about 55 Ps have 2 Qs, though not necessarily the same 55 Ps. There are 3 Ps with only one KS (M40, V10, and A77), and 20 Ps with only 2. There are about 10 Ps for each frequency class for 5 to 8 KSs in each of the N and Q classes, and the other KS frequencies are near 0 (S40 has the maximum of 19, with close runner-ups: C60, F75, F15, M55, M50, M30, M20, and F35).

With respect to principal KSs, a majority of KSs that are principal are principal for only one P. But only about 100 Ns are principals, so that some serve as principal for more than one P. One way this is possible is illustrated by NS11: it is principal for S10, S13, and S15, each of which elaborates a case of its use under different conditions. ND1 (and other ND's) are composite, defining a set of words to be members of the same word class at once rather than (unconcisely) making a separate statement for each membership


Content of KS
KS
P
Comment on P
First function
NS1
Second function
NS2
Third function
NS3
Fourth function, if not FV
NS4
Fourth function, if FV
NS5
Qword is FV
NS6
Delimiter is end of chunk
NS7
First chunk is left end
NS8
Next chunk starts after current
NS9
Period not in chunk
NS10
Initial scan is l-r, four funcs.
NS11
End of chunk, evoke splitting
NS12
FV always last in problem
NS13
Answer building at end of problem
NS14
S10 Initialize scan
S13 Scan and apply transformations
S15 Scan (FV) and apply transformations
S16 Age transformations
S17 Apply dictionary tags
S18 Apply dictionary tags (Age problem)
S20 New high precedence
S25 No precedence
S30 Lower precedence
S35 Apply FV chunking rules
S40 Delimiter chunk
S60 Detect FV start
S65 End of FV
S70 End of problem string

Key: Direct uses are solid lines, weaker interactions, broken ones.

Figure C.2 The mapping of NS's to S Ps

assertion. NF1 is not strictly a compound statement, but F5, F15, and F20 each use a subpart of it as their principal component.

We now summarize the ways in which the various kinds of KSs can be extracted from Ps, based on the experience with the full Studnt analysis. As in the above example, the Ns are determined: by combining the meanings of predicates; by comparing the LHS and the RHS, using common variables; by the occurrence of NOT in the LHS, indicating an excitatory interaction. Determining the exact content, however, of Ns and Qs does require some kind of collection of several cases of use, so that an appropriate generalization can be made, for economy of expression. Also it must be determined in a non-immediate way just which terms are to be handled by definitional sorts of KSs, and whether those definitions are Ns or Qs. But these considerations really only apply when the reading is started from scratch, and once the basic terminology for a PS is established, the determina'ion process is much easier. To determine the Qs of the definitional and erasure types is quite straightforward: erasure knowledge is based on occurrences of negated templates in the RHS, and definitional knowledge can be assumed whenever there is some gap between terms in Ns and predicates. To determine programming techniques, the following clues are used: presence of signals; ordering of signals in the RHS; presence of data that is elsewhere used in a particular way (Q28, Q42); particular type of predicate (e.g., Q16); re-asserlion (Q42). For the Zs, we have the following: order of the RHS; re-assertion; seemingly strange condilion elements, for instance P-specific ones. With respect to the use of RHS order in determining Qs and Zs, something more must be known than local considerations, since Psnlst does not have an explicit notation for which of the RHS elements really do have an important order relative to each other. This "something more" is simply closeness to the principal KS of the P, or closeness to the problem space plans that are directing the processing. In general, only the first few elements, or in most cases just the first one, have an ordering constraint, with the rest being don't-care's.

## C.7. A model of knowledge acquisilion

The process of knowledge encoding fits into a model of knowledge acquisition along the following lines. An artificial intelligence is seen as an entity with capability for gathering pieces of information, which are used in formulating behavior patterns organized as problem spaces. A piece of information by itself is insufficient to produce appropriate behavior. Rather, it must be assimilated or understood by having it fit into models that have been previously acquired or that are built up by a problem-solving process. This process of understanding consists of first expressing the new information in terms that overlap with some problem-space-level model and then allowing the information to interact as illustrated above to form new P rules. This broad model goes along with the view that intelligence is increased by increasing the ability to select a particular behavior out of all the possibilities in a given situation. In the PS model, selectivity is increased by adding rules and by correspondingly increasing the complexity of P conditions. This growth in selectivity can easily be seen as growth in a discrimination net (see Rychener, 1976, or Hayes-Roth and Mostow, 1975) in which each condition element is taken as a node in the network. A match to a P condition then corresponds to finding a path in the network to a terminal node, at which are stored the elements corresponding to the action side of a P.

Figure C.3 illustrates the components of the model. Each box in Figure C.3 represents some body of knowledge, either as an abstract model or as a specific set of detailed facts. Boxes in solid lines have already been discussed, along with the processing indicated by the arrows that results in the Ps. Boxes in broken lines are parts of the process that are hypothesized, but are insufficiently elaborated at present to permit further specification. The figure shows static data dependence; i.e., it indicates that knowledge in one box is used in forming the knowledge in the other. It doesn't indicate anything, for instance, about how a knowledge encoding process would access the various bodies of knowledge dynamically, nor does it include the knowledge extraction process. Except where arrows merge, interaction of knowledge (as illustrated in Section C.2) occurs within the boxes, e.g., Ns with other Ns. The arrows show, rather, how a body of knowledge forms by development or elaboration from other knowledge (e.g., box 4 to box 5), or how such developments merge in a largely additive way to form a body of knowledge (6, 7, and 8 into 9).

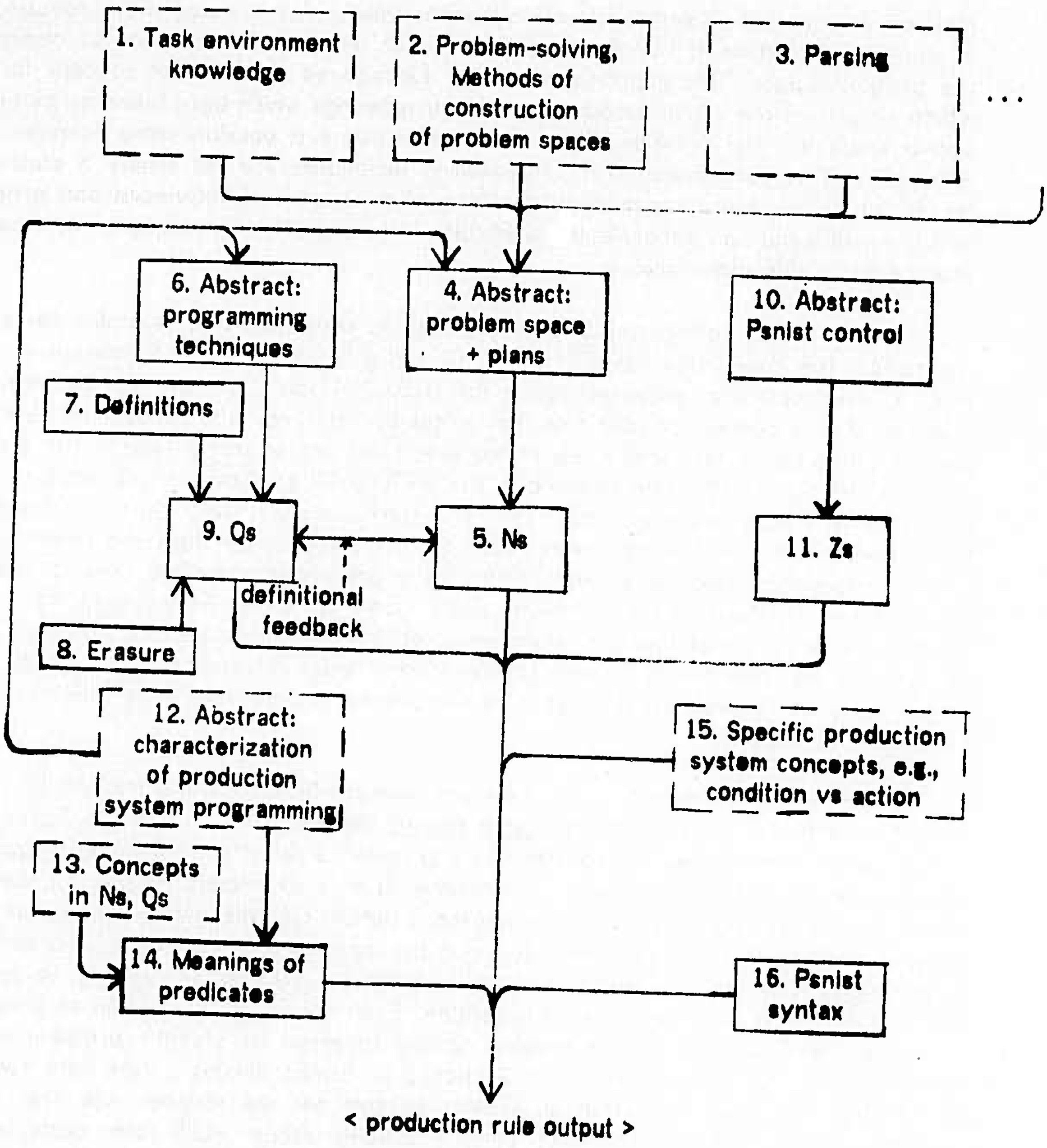

1. Task environment knowledge
2. Problem-solving, Methods of construction of problem spaces
3. Parsing
...
6. Abstract: programming techniques
4. Abstract: problem space + plans
10. Abstract: Psnist control
7. Definitions
9. Qs
5. Ns
11. Zs
definitional feedback
8. Erasure
12. Abstract: characterization of production system programming
15. Specific production system concepts, e.g., condition vs action
13. Concepts in Ns, Qs
14. Meanings of predicates
16. Psnist syntax
< production rule output >

Figure C.3 The model of knowledge acquisition: Bodies of knowledge

Some of the broken-line boxes are not expected to present much difficulty, namely 12, 15 and 16. The others represent more difficult problems than what has been solved so far. Boxes 1-3 are where much of the real high-level problem-solving takes place, namely in the precise formulation of the task environment and in the construction of the problem space within which dealing with that environment is possible. It is during that process of formulation and construction that the intelligence is added which results in part in the "plan" portion of box 4, that portion which directs the application of operators in the problem space. The specification of box 13 requires a process of concept-formation, which results in the set of predicates and their meanings which were taken as given in the above analysis. The creation of the elements in box 6 is possibly more complex than is indicated. It is conceivable that programming techniques are not simply a collection of facts, but rather are a capability in the form of more general knowledge and procedures which on demand can generate the particular instances of programming know-how which are the Qs in the above analysis.

With respect to Figure C.3 it only remains to point out some examples for a few of its parts. The connection between boxes 12 and 6 is unused in the formation of most of the Qs, and we have discussed above for Q15, Q24 and Q40 some of the problematic aspects of this connection, and how they might be resolved. The connection between 12 and 14 reflects the fact that a few of the predicates are oriented towards the structures used in the Psnlst PS. One example is the HASCPRIOR predicate, which assigns to each chunk in a Studnt problem a priority. If a stack data structure were available, these numerical values could be done away with, since the result is a stack-like ordering of the chunk processing. Another example is the set of predicates which are used to keep track of the tree structure of the arithmetic expressions. In a Lisp environment, for instance, the recursive nature of function calling would encode the same concepts. Finally, it should be pointed out that boxes 10 and 12 may have enough in common to be merged into a single body of knowledge, although with the present limited objectives their distinctness can be maintained.

The major component of the task environment (box 1) is the method to be used. Studnt's method is a variant of the Match method (Newell, 1969), where the "form" against which inputs are matched is expressed as a grammar, a set of rules capable of generating all possible forms to be matched. The grammar itself is not implemented as a generator of forms (top-down) but rather as a recognizer, a bottom-up precedence-based parser. The transformations that Studnt applies to bring the input to a recognizable form correspond to normalizations that are sometimes done by template matching procedures, to get inputs into suitable form for a given set of templates. Even if we take the method as given, there is still a significant amount of problem-solving to arrive at Studnt's problem space as described by the abstract model in Section C.1. Studnt divides a task into two parts: processing the input to arrive at a form suitable for the matcher and the matching (parsing) itself. To get the first part, a problem-solver must form such ideas as: transformations on strings; classes of words; marking word classes with tags; organizing the process as a left-to-right scan; organizing the input string as a series of chunks with delimiters and operators as boundaries; and so on. The match has two distinct components, the parsing process and the variable-identification process. The parsing uses: the concept of chunks; the system of operator precedences, which must be extracted from ordering relations noted in the task environment somehow; properties of FV-specific words; and so on. Studnt's variable-identification process, which is applied after a structure has been parsed, is not itself a parser but consists of a rather weak collection of equivalence rules, but even this rudimentary process uses: a left-right scan of variables to be identified; rudimentary pronoun referent substitution; and specific equivalence rules.

The phrases above referring to left-to-right scan bring out once again (cf. Section

C.1) another feature of the requirements of problem space formation: the addition of plans. Plans take the place of exploratory (backtracking) search, so that their appearance in a solver's problem space is of importance. It remains a significant problematic aspect to determine how they're added.

To summarize, the problem-solving involved in forming the problem space is of an ill-structured nature, requiring concept-formation and plan-formation processes that are poorly understood at present (but see the discussion below in Section C.9 of the work of Hayes and Simon, 1973). Because the problems in studying the problem space formation process in more precise terms appear formidable, we should look for supporting evidence, and in particular we can question two aspects of the Studnt problem space: is it the correct problem space and can it be arrived at by other means. Concerning the correctness, there are three viewpoints: the human problem-solving viewpoint, the AI program viewpoint, and the implementation viewpoint.

The first view deals with whether there is support for the model from human problem-solving studies. Paige and Simon (1966) considered exactly this question, and their conclusion was that humans' basic problem space is like STUDENT. They went on to consider informally a set of augmentations of the basic problem space, suggesting that STUDENT could accommodate at least some of those augmentations. The Paige and Simon paper did not consider the protocols relating to the basic problem space in sufficient detail to support or contradict the finer details of the STUDENT model, such as its system of operator precedences, but it is safe to assume that no gross differences were evident.

The AI program viewpoint considers the question of whether Studnt (or STUDENT) can be extended comfortably to the real task, namely problems chosen without care to simplifying the language. My informal examination of a set of 33 problems from a college-level algebra text (Rosenbach et al, 1958) can be summarized as follows: none of them are directly solvable, five could be solved by easy extensions, 14 by harder extensions, and 14 by extensions of major difficulty. By easy extension, I mean addition of simple idiomatic transformations. By harder extension, I mean adding specialized knowledge to solve problems in particular domains of discourse, such as problems dealing with coins, interest, and mixtures (chemical solutions and alloys), and adding more context dependence to certain idiomatic transformations and pronoun referent replacements. By extensions of major difficulty, I refer to: problems requiring elaborate semantic models to create the set of equations, that is, where some inference is required to derive necessary relations from given information (e.g., certain complex rate-distance problems, for which a diagram is an essential part of a human's solution); problems requiring elementary knowledge of points, lines, and curves; problems calling for symbolic solution as opposed to numeric; problems requiring solving a previous problem with different numeric values; and problems requiring operations on relations, such as reversing the role played by two variables. This last class of extensions also has the property that a problem solver that is an extension of Studnt would spend more of its computing effort in the extension than in the basic Studnt mechanisms. This is not the case, I believe, for the first two classes of extensions. This assertion can be supported by results obtained with respect to examining the age-problem heuristics as an extension of Studnt, within the present Studnt, which is discussed in more detail below in considering the extension as an addition of KSs (Section C.8). The age-problem extension is of the harder extension category. From this breakdown of how Studnt might be extendec, we can take some support for the present problem space formulation.

The implementation viewpoint concerns itself with the problem of implementing the given version of Studnt, which has been solved in (at least) two cases. If the model of formulating the problem space, given the task environment, and then encoding that problem space as a program, is approximalely correct, then the problem solving involved is of a particularly high order, especially in comparison to the state of the art in AI. But since it is likely that the conceptual structures we find in the finished programs correspond to the problem space organization that aided in their implementation, we have still further support for the correctness of the present formulation.

This last topic ties in with the second aspect of the problem space formulation that we might question, namely whether the given problem space can be arrived at by some other means. In particular, can it be arrived at by a simple specialization process on previously-learned natural language processing? Has simplifying the input domain and building up a problem-solving process from scratch added unnecessary complexity? Given the lack of evidence on this, in parlicular with respect to more capable AI programs, we can only offer a few speculations, remaining within a human problem-solving viewpoint. Perhaps humans, in solving this class of problem, do not rely on plans as much as on weaker search-like methods. Thus the plan-formation aspect of the problem space formation process may not need to be explained. It is necessary, in addition, to consider the role of teaching and imitation as aids in the process (and perhaps teachers and authors of texts could benefit from the AI formulation). But certainly the concept-formation process is only pushed temporarily out of sight by saying that the problem space used is a specialization of some familiar capabilities. That is, the concept formation took place somewhere during the arising of these capabilities, although its occurrence over a longer period of time may make it, ultimately, more easily explained.

## C.8. Conclusions on the knowledge analysis

The knowledge analysis has shed light on the essential aspects of how knowledge is encoded in PSs, and thus takes a definite position on how PS programs are written, augmented and refined. A PS program starts out as (partial) encoding of knowledge stated in terms of some problem space. Ordinarily, the program is then tested, and defects come to light as a result of interactions that were not considered in the original encoding. The new interactions may be deall with by forming new KSs which are then considered as additions, or they may correct oversights in processing that produced the original. For knowledge to be added, it must first be stated in terms that make contact with the problem space in which the program is formulated (or with an abstract model at the problem space level). Then there must be consideration of the ways the new piece of knowledge can interact with the given ones. In determining those interactions, the explicitness of expression, allowing knowledge content to be easily read as explained above, is instrumental. Replacement or modification of knowledge requires a similar consideration of interactions. It is important to emphasize that in this formulation, program behavior can not be augmented by simply adding Ps, as is the case in some rule-oriented systems, because Ps here are encodings of more than one KS. This is the case because of the conceptual structuring provided by the problem space (model). The circumstances allowing simple addition of rules are those where the plans in the problem space are lacking, so that some method of heuristic search among possible behavior sequences is undertaken. This allows the addition of knowledge in its pure form because at the higher level there is very weak structure, and no basis for determining any interactions.

The analysis has demonstrated the directness of encoding of problem space knowledge, by virtue of the ratio of Ns to the other types of KSs. With 154 Ns, 53 Qs, and 11 Zs, it is apparent that the Ns predominate, and that the control knowledge specific to PSs is quite minimal. These figures do not include the very low-level PS syntactical knowledge, for two reasons. That knowledge is fairly constant over the entire set of Ps, and it is sufficiently simple that it quickly becomes automatic for the programmer, requiring little attention during the programming process. Not only is the encoding of knowledge direct, with little knowledge required to bridge the gap between a high-level problem space description and the actual language, but it is also the case that the size of each programming unit is small in terms of number of KSs: on the average, each P contains 2.88 Ns, 2.86 Qs, and 0.65 Zs. It is asserted here that the above analysis indicates that little other knowledge needs to be considered beyond these 6.39 KSs. The explanation for this is that the structure of the problem space has satisfactorily co-ordinated its component KSs.

Because, with this framework of encoding, it has been possible to consider knowledge at rather general levels, it is appropriate to view it as the beginning of a comprehensive model of knowledge acquisition. It takes an explicit position on what knowledge is (at the natural language level, but not at a more formal level), it proposes mechanisms for its incorporation into some existing body of knowledge, and it exhibits the result of assimilation of knowledge, namely the Ps. It is interesting to point out that other experiments have indicated how P conditions can be stored as an EPAM-like (Feigenbaum, 1963) discrimination network (see Hayes-Roth and Mostow, 1975, Waterman, 1975, and Rychener, 1976). The present formulation also indicates how processes of problem-solving and concept-formation enter into knowledge acquisition. It takes a clear position on the difference between knowing and understanding some piece of information, namely that knowledge is not understood fully until its interactions with other knowledge have been considered according to the knowledge interaction process hypothesized here.

As a model of knowledge acquisition, this approach may contribute to the automation of learning or of incremental addition of knowledge to a PS program. Going further, it may suggest a different mode altogether of expressing PS programs, namely natural language (or at least some language that expresses knowledge in a way similar to the KSs, orthogonal to the Ps), and in a more limited implementation, would constitute a powerful "programmer's helper". Along these lines, it can be noted that the division into Ns, Qs, and Zs would perhaps remove the burden of specifying programming techniques from the programmer. Also, variations in programs would result from variations in the set of predicates used by the program in constructing programs. That is, the predicates form a conceptual base for the programming system to work with, which might best be determined interactively.

The three subsections that follow contain some tentative conclusions from this work, and attempt to structure its extension, its development, and its application to other areas. First, we compare this approach to related work and point out how this approach might be used to restructure those results. Then we consider Studnt as an understanding system and propose some ways that a knowledge encoding analysis can be used to measure various dimensions of understanding. Finally, further research that is essential to supporting this analysis will be discussed.

## C.9. Comparisons with other approaches

It is difficult at this time to compare our results with other approaches to encoding knowledge, because no other studies have taken a sufficiently similar approach. However, we can point out features of interest as viewed from this approach, and indicate further studies that might be undertaken to this end. The reader is cautioned that some topics are raised in a very cursory fashion, with the intention that these may deserve further consideration based on this initial exploratory examination. This subsection is primarily intended to sketch how this work seems to relate to other approaches.

A very interesting comparison can be made to another PS organization, Newell's (1973) PSG. This comparison is based on thorough knowledge of that system, but not on a detailed implementation of some program in PSG. The commonality of PSs indicates that we should only have to look at the corresponding Zs. PSG is a PS interpreter in which Ps detect conditions in a linearly ordered Working Memory (STM). As a result of detecting conditions, specific actions are performed, consisting of adding, deleting, modifying and re-ordering the elements of STM. When more than one P condition is true at the time of recognition, that P is allowed to fire which uses STM elements closest to the front of STM. The detailed comparison is as follows (cf. the Z model given above, Section C.5):

a. Order in RHS and order of examination of Ps: very similar to Psnlst, except order in the RHS is reversed; in PSG, the last (rightmost) RHS insertion is at the front of its STM.
b. Re-assertion in Psnlst corresponds to data rehearsal (the NTC action) which brings elements to the front of STM.
c. Matching and the problem of spurious P firings: it is possible to put elements in front of other elements, so that the others don't take part in matching, but PSG has no new-old distinction on STM elements; thus some (ad hoc) unknown memory structuring must be used to prevent spurious firings (e.g., renaming data elements, which retains the information but changes the set of sensitive conditions).
d. Problem of contradictory actions: either non-existent because of the order of actions, with deletions getting done before insertions generally, or it must be handled in the same way as in Psnlst.
e. The control of looping is the same for both systems.

This comparison of PSG and Psnlst does not deal with all of their differences, because it is limited to the control mechanisms only, and because the control mechanisms that have to be considered are limited by the domain determined by Studnt. Our conclusion is that PS control issues are essentially the same in both systems, increasing our confidence that our assertions about PSs have some general validity.

With respect to more conventional languages, a couple of points can be made as motivation for more detailed studies. The step size of PSs compares quite favorably to a small recursive LISP function. That is, a P and a recursive lambda expression have similar size, expressive power and isolation in terms of knowledge content. LISP, however, generally suffers from the "subroutine interaction problem", since knowledge interactions are not carried through to the extent allowed by PSs. The size of programming unit is much smaller than an ALGOL block structure, where the assumptions at some point in a program depend on a lexically very large extent, with each inner nested block inheriting knowledge assumptions from its outer containing blocks. If we were to attach assertions at various points in an Algol program corresponding to KSs that are assumed at those points, then places that are nested in several block levels would have all the relevant local assumptions plus those of all the containing levels. For a P, the KSs that hold are determined locally. Thus a PS program has knowledge distributed more or less uniformly over its parts where an Algol program shows wide variations in density of knowledge. Proving correctness of a conventional program is done by attaching assertions to a flowchart and then following the flowchart sequentially, verifying assertions at a point in the context of accumulated assertions from the flowchart traversal, whereas for a PS, verification can be (it is claimed, to be supported by further research) much more localized, with no need to deal with control flow. The knowledge encoding approach poses the question of proving correctness of programs as the process of determining the following features: the knowledge content; whether the knowledge is correctly encoded, i.e., whether all relevant interactions have been explored; and whether the knowledge is correct with respect to the given task environment.

If we are to compare PSs to Planner-like languages (see Bobrow and Raphael, 1973) it is essential to point out that at the Z level, these languages have a pattern-goal-oriented implicit search, which may have large ramifications on how the other knowledge levels are formulated. A more general question to be answered is how the encoding of knowledge as Planner theorems is different from encoding it as Ps. An attempt at making a system flexible in terms of augmentation was done by Winograd (1972), and the result (unpublished) was that to add certain kinds of knowledge, some other knowledge of the internal workings of the program was necessary. In other words, more than just an N-like statement, with pure problem space content, was necessary. Charniak's (1972) systematization of a body of knowledge relating to children's stories would have to be re-formulated from a problem space viewpoint, in order to make comparisons. This is made more difficult because there is a lack of explicit statement as to exactly what that body of knowledge consisted of. A good deal of discussion by Charniak was based on the body of knowledge without getting down to a strict separation of the knowledge from various interesting issues related to it.

A recent study by Hayes and Simon (1973) investigates the process of extracting problem-space-related knowledge from the instructions for a problem-solving experiment. This involves studying protocols of human behavior, and attempting to model the processes as a computer program. The program assumes a particular form for the problem space: the GPS (see Newell and Simon, 1963) form of heuristic search with means-ends analysis. The program thus addresses the area dealing with boxes 1 and 2 in Figure C.3; its output is a set of task environment "statements" that have a form suitable for input to a GPS-like problem solver. Although the work covers only a small portion of knowledge acquisition as outlined above and makes strong assumptions about the desired form of the problem space, it serves as a useful base point for further work along the lines of the acquisition model and especially for the problem space formation process.

Finally, we compare the present approach to Sussman's (1973) model of skill acquisition. The model (Hacker) deals with the knowledge used in constructing problem-solving procedures in a toy blocks world. There are several categories of Hacker FACTs (its version of KSs): one deals with details of the toy blocks world, giving attributes of pre-defined blocks operators, for instance preconditions for the PUSH operator; a second is programming techniques, which deal with the particular model of problem-solving being used; two others deal with Hacker's "critics' gallery," a body of knowledge about bugs in procedures; the fifth is a program library, with procedures that have been used for previous problems; and the sixth, a "notebook" with comments on programs in the library. Faced with a problem, Hacker uses the appropriate bodies of knowledge to build a first approximation to a procedure to solve the problem. By executing that procedue in a "careful" mode, bugs in the procedure are uncovered, the critics' gallery proposes a solution, and another attempt is made to execute the procedure. An example of how this works treats the problem of writing a procedure to build a tower of blocks. An initial procedure might do fine until it tries to pick up a block with another block on top of it. At that point, the critics' gallery might propose inserting a line of code before the pick-up operation, to ensure that no block is on top of the block to be picked up.

Sussman did not gather together his FACTs and comment on them as a group, but by my count, there are 12 blocks-world FACTs, 16 programming techniques, and 8 critics' gallery FACTs. FACTs relevant to a particular procedure are not all combined at once as envisioned in the present knowledge encoding process, but it is necessary to put together a first approximation to a procedure and then execute it to see what happens. Thus, it is not the case that knowledge can be extracted from Hacker's procedures by an analog of our knowledge extraction process. To find out in detail the properties of a library procedure that was constructed previously, in order to modify or generalize it for a new problem, it has to be executed and its behavior monitored. Also, if the result of careful execution is new knowledge, that knowledge is not incorporated into the procedures for generating programs, so that it would be used appropriately for future problems, but rather it becomes a new entry in the critics' gallery, and can only be used to patch up bugs in carefully-executed procedures. In principle, there seems to be nothing to prevent the critics' gallery from growing to very unmanageable and inefficient proportions, especially with the possibility of critics' being formed to correct other critics' actions.

Sussman's Hacker approach takes a definite and more or less traditional stand on the issue of modularity of knowledge, whereas the proposal here represents a different approach. Hacker's KSs are kept in pure form as FACTs and grouped conceptually into modules that are claimed to be substitutible or interchangeable for modules dealing with other problem domains. The Studnt approach (ideally, given that the present model can be implemented) is that individual KSs are not kept in pure form but only in the encoded form. The encoded form, however, is sufficiently explicit that the statements can be recovered, at least enough to compute further interactions. Modularity is still maintained at the problem space level. Thus the PS trades explicitness of representation for individual statement modularity. Just how the PS approach as proposed here would be worked out in detail is still an open question, and will be discussed below in connection with problems for further research.

## C.10. Understanding and intelligence in Studnt

In order to approach issues related to understanding, intelligence, generality, and similar topics, we adopt the understanding dimensions approach of Moore and Newell (1973). Moore and Newell define understanding by saying that a system understands some piece of knowledge if it uses it whenever appropriate. They propose eight dimensions along which understanding systems are to be evaluated: (1) representation of knowledge; (2) action, the conversion of knowledge into behavior; (3) the assimilation to the internal structure of external (task environment) structure; (4) the accommodation of the internal structure to external structure (which includes learning, incorporation, or acquisition of new knowledge structure); (5) directionality, the structure that initiates and guides processing toward specific ends by appropriate use of knowledge; (6) efficiency; (7) how the system responds to error; (8) depth of understanding, an indication of how effectively knowledge is brought to bear whenever appropriate. Studnt takes Ps as its ultimate knowledge representation, and the interpretation of Ps as the means of obtaining action. The following paragraphs discuss (3) through (8) in turn.

Assimilation will be posed as a question of whether Studnt adequately encodes all of the KSs. That is, a KS is said to be assimilated when its encoding in Ps has been effected. For Studnt, this question is mapped into determining whether all of the interactions of the knowledge have been correctly considered. Evidence that the program can solve problems that require various subsets of its KSs is at best only indirect support that it understands the knowledge. We must postpone a definite determination of Studnt's degree of assimilation of its knowledge until more concrete progress is made in automating the knowledge-encoding process, thereby making more definite the meaning of interaction. The best possible estimate at present is based on taking the number of uses of KSs in Ps as the number of interactions (roughly 1650) and taking the number of "bugs" discovered in the process of the analysis (about 50), to get 97%. Even though this figure is suspect because it relies on the accuracy of my own judgment as to what is correct for the 50 bugs found (in general, a more knowledgeable encoder is necessary, to judge the result of an encoding process either directly or through behavioral tests), it illustrates a measure of assimilation based on the knowledge encoding approach.

The accommodation dimension raises questions with regard to how the Studnt structure can be augmented to expand its area of performance. As discussed above (Section C.5), on a sample of 33 problems not given to Studnt, 5 (15%) would require easy extensions⊗ to Studnt, 14 (42.5%) require harder extensions, and 14 (42.5%) require extensions that present major difficulties. The first two classes (57.5%) could be reasonably said to be within the range of Studnt's ability, while the rest require such radically different approaches as to be beyond Studnt, in the sense that the "Studnt" nature of a program to solve them would be diminished relative to the total program. Thus Studnt might be said to be 57.5% accommodating. These figures are, of course, based on this author's judgment of problem difficulty. They are suspect also due to the sample chosen: it is indeed a significant problem to determine what set of problems to examine. Studnt can solve a class of problems of unlimited size, and there are classes outside its reach that are also unlimited. The cautious appraisal of the 57.5% figure would be that it illustrates a possible methodology for measuring accommodation, dependent upon the knowledge-encoding approach (as it is used to evaluate the knowledge necessary to effect the accommodation), but that a great deal more research is necessary in order to support both the general approach and the specific measurement obtained.

⊗ Perhaps easy extensions are more properly considered to be assimilation, since they require little structural change.

In keeping with the statement at the beginning of Section C.7, I would identify directionality with intelligence. This accords with the view expressed by Newell and Simon (1972, chapter 3, pp. 88-89) that the intelligence of a problem-solver is related to the difficulty of a problem for that solver, as evidenced by its search behavior. That is, the more directed the search is, implying examination of fewer irrelevant alternatives, the more intelligent the solver. Studnt's intelligence cannot be measured by examining its search behavior, because the only sense in which it does search is that it constantly progresses toward completion by scanning, chunking, and building expressions. We can, however, examine qualitatively the knowledge that directs the constant progress, and comment on how it might be possible to formulate its limitations by studying the space of problem spaces. Studnt's intelligence is embodied in the plans it uses. These plans are inflexible, prescribing specific actions in specific orders. According to the model of knowledge acquisition presented above, this intelligence is acquired during the problem space formation process, and if the intelligence is limited, it is due to limitations in the problem space. As Newell and Simon point out, if the problem space were richer, allowing the direction of processing to be based on more appropriate discriminations (as required by the task environment), a problem solver (Studnt) would have greater potential intelligence. A more exact understanding of the space of problem spaces for solvers of Studnt-like problems might allow Studnt's intelligence to be measured relative to other programs. Such a measure might be based on an analysis of knowledge in the form of plans contained in such problem solvers, especially if the body of knowledge formed by taking the union of all such sets of plan knowledge is a coherent whole.

On efficiency, the main point we can make is that since knowledge is encoded procedurally as Ps, with only the temporary state in Working Memory, the interpretation sub-issue has little impact* The interpretation sub-issue is that if many levels of interpretation of knowledge are required, the factors of extra computing time required at each level multiply (cf. the difference in running a program compiled, interpretively, or on a simulated computer). In particular, while Studnt is solving a problem, it is not the case that it must search to find the implications of some piece of knowledge or to decide how two items of information must interact. This apparent efficiency is at the cost, perhaps, of an expensive knowledge encoding procedure; this cost will only be known after further research.

In the general category of error, the knowledge analysis leads to the consideration of how to assign blame to particular KSs for some faulty behavior. This approach says that the error is not localized in particular Ps but rather is due to faulty (incorrect) KSs or to failure to consider interactions between KSs; thus an error may be due to the contents of a set of Ps. In diagnosing and correcting an error, it is clear that the processes of knowledge extraction and knowledge encoding are essential. We can speculate that not only will the contributing KSs have to be known, but that some relative reliability measure on KSs might be useful (reliability perhaps determined by successful use on past problems), in deciding on corrective action. For the present Studnt, there is a computation of the contribution of particular KSs to the total behavior. The listing of the KSs, Appendix F, gives the Ps in which each KS is used, and the actual TESTs in which each KS is applied by virtue of some P, which incorporates it, firing during the TEST. For instance,

* We will ignore whether Ps themselves are interpreted or compiled, given the understanding-system level of this discussion.

it is clear that the almost all of the NS's (initial scan) are used for all the TESTs, whereas each of the NT's (transformations) is used in a small subset, where the subset varies according to which NT is examined. On the whole, the NT's, the ND's, some NM's, some NC's, the NF's, the NA's, 14 of the Qs, and Z8 are used only in subsets of the TESTs, while the other Ns, 39 of the Qs, and the Zs (except Z8) are used in all of the TESTs.

To measure Studnt's depth of understanding within the knowledge-encoding approach, it is necessary to consider whether all knowledge interactions are properly worked out. For instance, it might be possible to construct an example that uses knowledge in Studnt in such a way that Studnt fails to apply it appropriately. Such an example has not yet been found, but that doesn't rule out the possibility entirely. (This task is much more difficult than finding problems that use knowledge that Studnt doesn't have at all, or finding problems where Studnt's knowledge is inaccurate.) The kinds of interactions that are worked out are perhaps determined by the problem space, so to find a proof or counterexample, it may be necessary to have an exact and full understanding of how interactions are related to the problem space (more is said on this in the following subsection).

## C.11. Directions for further research

The analysis of the knowledge in Studnt has provided a framework for posing further research questions relating to four major areas: (1) verifying the analysis by automating the knowledge-encoding process; (2) testing the extendibility of the model by adding knowledge that extends the domain of solvable problems; (3) testing the substitutibility of the model components by trying to apply the analysis to other programming languages; (4) testing the applicability of the overall model of knowledge acquisition by similar analyses of AI programs for other task areas. We have already presented some directions to go on question (4), in Section C.9. Topics (2) and (3) depend to a large extent on progress with respect to (1), either using PSs or some other programming language. The following paragraphs speculate on the central issues to be resolved in attacking question (1).

An immediate question relating to automating the analysis is the choice of language for the KSs. One approach is to analyze the KSs themselves for underlying semantic structure, in order to determine the kind of mechanical translation that needs to be done to express the knowledge in a directly assimilable form, or in order to design a more suitable formal notation. Natural language was sufficient for the purposes of the present first approximation at a model of knowledge, and its use obviated the need to do a design of a formal language at the same time as the analysis was being done. Certainly it is not necessary to have a language more powerful than natural language, but rather it may be necessary to use a language that places less burden on the processor in filling in implied relations and objects. Any use of an artificial or formal language faces another problem: how to guarantee that the formal language has a systematic basis, or that it is possible to decide how to express some idea, for instance with or without making ad hoc extensions to the language. Sussman (1973) and Charniak (1972) both expressed knowledge in formalisms directly usable by their (partially hypothetical) programs. But they in fact ignored the theory of construction of these formal assertions, and in many cases simplified and altered them for human readability. (These two are emphasized in preference to "pure" predicate calculus formulations for the reason that the predicate calculus approach has not been practically applied to such task areas to date.) In other words the systematization of expressing the knowledge is inside the head of whoever is using it and is thus for purposes of analysis effectively lost. Also the parts of the programs that make assumptions about input form are scattered, rather than collected into a language interface. Using natural language, on the other hand, necessitates building some translation program, but that program can then be inspected, presumably, and the theory of construction of formal representations of knowledge that it embodies can be extracted and made explicit.

The analysis of the KSs, either with a view towards using an artificial language for further work or as the actual interface to the encoding process, will require advances in the present state of the art. The most promising approach at this time may be to use ideas similar to those of Hayes and Simon (1973). Their approach, which was successful in analyzing the task instructions for a problem-solving experiment and which derives from an approach to automated protocol analysis (Waterman and Newell, 1973), is based on loosely processing the natural language input, attempting to make connections with known forms, but otherwise ignoring parts of the input that cannot be parsed (the parser is designed to react flexibly to such noise).

As an adjunct to the actual automation of the process, it might be useful to test how much of the scheme can be used by humans in writing PSs. It is reasonable to look for a strategy of making explicit the knowledge to be encoded, at the same stage in the programming process that is occupied by a top-down "structured programming" strategy with a more conventional language. This would divide the programming into two stages, one involving the clear formulation of the body of knowledge to be encoded, and the other involving the problem-solving necessary to complete the PS encoding.

The representation of the KSs internally is another major unsolved problem. The main aspect of this is the question of duality of representation: is it necessary to keep both the procedurally-encoded knowledge as it exists in the Ps, and something corresponding to the individual KSs? It seems essential that knowledge be kept available for interactions arising some time after its initial acquisition. A fact might even be made use of for constructing and revising many different problem spaces, in addition to aiding the addition of knowledge in closely related areas. As sketched above, it seems plausible that a program could determine the knowledge in a P by examining it, given the meanings of the predicates, and given an overall understanding of the problem space. It might be possible to aid this process considerably by encoding the P LHSs as a discrimination net, and then using the net to discriminate, and to study the interactions of, the KSs themselves. Thus the net would simultaneously represent the desired duality, with one interpretation being used to match conditions of Ps, and another interpretation, based on predicate meanings, to regenerate the knowledge content of Ps. This adds to the design considerations for representing Ps as a discrimination net, and provides more motivation for pursuing that topic further.

Several questions can be formulated with respect to the various components of the above analysis. First, it might be necessary to refine the decomposition into Ns, Qs and Zs that was developed above, since automating may add requirements to the structuring of the statements. The process of determining which KSs are to be taken as principal ones needs more exact specification. It might be fruitful to investigate the question of how to generate the predicates, which would involve trying to characterize predicate meanings in a general way, as well as the question of how to refine this concept structure to fit the needs of the specific implementation. This aspect would involve, in advanced form, the examination of the Ps' structure to determine which subsequences of conditions would be more suitably expressed as single predicates, perhaps making decisions as to whether some predicate could be computed once instead of being recomputed on demand, or vice versa. Finally, the question of whether Qs need to be kept as a body of statements (either explicit or implicit, depending on the solution of the duality problem) or whether there might be some method of generating techniques from more abstract statements, by some kind of problem-solving process with knowledge of functional aspects of programming.

The process of how the KSs interact to form the Ps needs to specified much more carefully. Particularly important is to break them down in such a way that their associations and inter-relations with each other are clearer. The knowledge about Psnlst syntax at the lowest level, which wasn't considered here, would probably be encoded directly in the P-building processes. The process of applying the KSs of the Q and Z type requires recognition of conceptual structures that are not well understood at present. For instance, there would be a general set of criteria for recognizing a situation where knowledge about looping techniques can be applied (some of these situations are explicit in the Qs at present, but the statement of a general set of them, and how they're applied, remain as open problems). How the Ns interact raises the most interesting questions, which are difficult to approach at the present informal stage of the analysis The model for the Ns (or the problem space that it represents) seems to provide a rich interconnecting structure for the basic objects that are described by the Ns. This structure allows some kinds of interactions and development to take place, and prohibits others. For instance, the model makes a clear distinction between chunks that represent arithmetic expressions and chunks that represent the find-variable (FV) specifications for a problem; processing done on arithmetic expressions is by this distinction determined to be unnecessary on FVs. Since this kind of dependence of interactions on the containing model (problem space) structure was not central to the analysis of Studnt, it may be that it begins to have important effects only on more complex task domains, but it may be that the dependence will become evident as the analysis is automated.

Further research must be directed towards supporting the idea, implicit in the formulation of the knowledge acquisition model, that knowledge can be compartmentalized in various models. One interesting problem is to make explicit the model of pure task environment knowledge (box 1 in Figure C.3), and similarly another is to produce a pure formulation of the problem-solving methods. The use of models to replace the loose abstract descriptions provided for the Qs and Zs (Section C.5) is an important topic to pursue. The Q model must include functional goals like flexibility and efficiency, which are evident in some of the Qs, but which are at present isolated and unrationalized attributes.

The higher-level components of the model of knowledge acquisition, dealing with the formulation of the particular problem space given the nature of the task environment, introduce a very interesting set of research problems. As detailed above, there may be a significant amount of problem-solving and concept-formation in this process. This involves, for instance, the recognition that arithmetic operators form boundaries for portions of text, and that the operators can be processed by techniques used for phrase-structured grammars. Given some weak-method formulation of the problem space, such as some way of using heuristic search, the addition of the problem space plans used above constitutes an interesting learning problem.

The relationship of PSs to the overall knowledge acquisition model needs to be empirically determined. That is, a convincing case needs to be made that PSs can adequately represent the wide variety of procedures and data that have historically been used in AI programs. For instance, can PSs be used to represent semantic networks, and inferences of the type that have been achieved by using backtracking search? On a more general level, it would be useful to characterize the varieties of knowiedge, and how knowledge is encoded and manipulated, for the full range of past AI systems. It may turn out to be the case that the class of programs whose knowledge fits into the present framework is limited. Whether this is the case might be determined by analyzing other PSs using the present methodology. A particular area of current interest is the problem of representing uncertainty of knowledge sources (Shortliffe, 1974) and of learning and generalizing from real environments (Becker, 1973). At one level of description, more generally applicable Ps are ones with more general condition elements, but the process of acquisition and creation of more general knowledge for forming those elements needs a great deal of elaboration.

The present analysis has tried to elucidate as many aspects of the knowledge encoding process as possible, without becoming committed to an amount of further work that would be impossible in the scope of the present paper. The fact that the analysis includes details for the entire Studnt program supports the basic conceptual structure of the model, and allows certain important conclusions to be drawn about how knowledge is encoded in PSs. It is suggested that this level of detail is appropriate for the other studies of knowledge encoding outlined above. Further detailed research into the effectiveness of the model for use in an automated knowledge system is best postponed until more basic questions with regard to the use of PSs as a language have been investigated (see Rychener, 1976).

## D. Summary of Conclusions

Our conclusions from this study can be separated into those from the implementation itself and those from the knowledge analysis. Studnt adequately solves 27 tests that were done originally by STUDENT. Interesting features of program control as achieved by the PS are: the use of explicit data as control signals; the use of data elements to imitate a recursive (hierarchical) parsing of the inputs, and to build the tree-structured output expressions; the use of Psnlst's :SMPX to sequence and coordinate processing; and the use of Psnlst's multiple-firing capability in processing sets of items. The internal Working Memory representation of Psnlst embodies a choice for generality as opposed to the conciseness and ease of manipulation of a special-purpose string representation. The Working Memory is at least an order of magnitude larger than other known PS architectures can handle efficiently. The time efficiency of Studnt is quite reasonable for an interpreted language, and is less than an order of magnitude slower than a human on the same task. Studnt differs from STUDENT in the gross organization of the processing, doing a single left-to-right scan over the input to achieve what STUDENT did with several sets of rules applied in sequence, each of which made multiple scans of the input seeking various patterns. The two implementations use roughly the same number of rules, with Studnt's rules having more complex conditions and actions due to the data representation.

The primary aim of the knowledge analysis is to examine in detail the knowledge in Studnt and how it is encoded in the Ps. The knowledge is expressed as 218 natural-language statements of three broad categories, with the concept of problem space forming the organizational structure of the category comprising the majority of the statements. Each of the three classes of KS is described by an abstractly stated model, for which individual KSs are instantiations of detail. The S13 example illustrates the nature of the interactions of many knowledge statements in forming one of a set of related Ps. The mapping between Ps and KSs is many-many, due to the number of actions performed conveniently by a single P and due to the convenience of expressing KSs economically. This economy is in the sense of being usable for interaction in a variety of ways, thus gaining more contribution to the total Studnt program per KS. Data on the distribution of KSs over the full set of Studnt Ps give further support for the size of knowledge unit chosen and for the many-many nature of the mapping. An average P is the result of combining 2.88 KSs of the problem space type, 2.86 task-independent programming techniques, and 0.65 statements dealing with PS control. The mapping between problem space and Ps is fairly direct, given that of the 218 statements used, only about one fourth are programming techniques, with 5% of the total dealing with PSs. Thus the encoding process deals mostly with the addition of problem space knowledge. A brief look at a case of augmentation within Studnt indicates that most new knowledge is of the problem space category, with large overlap in the other categories. The knowledge analysis was developed entirely from the explicitness of P conditions and actions, allowing the knowledge to be read off in a systematic way.

The form of the knowledge analysis led to the hypothesization of a more comprehensive model of knowledge acquisition, as might be realized using PSs as a basis. The major problem of the formation of problem spaces from less structured task environment knowledge can be formulated in this model. This involves advances in the state of the art in problem-solving and concept-formation. Within the model, the process of programming in PSs is seen as a knowledge-encoding process, where the explicitness of PSs is used to advantage in debugging and augmentation. The decomposition of the knowledge into problem space versus programming techniques is promising in terms of being able to build up a set of standard techniques which would effect the encoding of numerous problem spaces of diverse sorts, amounting to substitubility of the various knowledge models. The utility of the model is based on being able to automate the knowledge-encoding process, which depends on being able to process the natural language statements, determine the knowledge content of existing Ps, and carry out the interaction process. The model thus raises numerous questions for further research. Techniques being developed in protocol analysis and in aspects of human understanding, exemplified by the work of Hayes and Simon, may provide a basis for the natural language processing involved.

Comparison to other approaches, especially Sussman's Hacker model, brings out the position of PSs vis a vis modularity of knowledge. The models of the KSs are modular, but the PS encoding is an explicit representation of the full extent of possible interactions among the statements. Thus the encoding is at the extreme position of a modularity dimension, with access to the knowledge in a modular way dependent on explicitness.

There are several benefits from positing a level of knowledge between its expression as knowledge about a task environment and its expression as Ps. KSs as exemplified here are closer to problem-space-level models than are Ps. There is significant problem solving, namely finding the interactions of KSs, in making the translation from KSs to Ps. There is also problem solving, of a different sort, in forming the problem space from knowledge of the task environment and knowledge or methods. The separation of problem space knowledge from programming techniques and lower-level PS knowledge is promising with respect to applying known techniques to new bodies of problem space knowledge, with a minimal need for re-shaping the problem space to fit the available techniques.

Measures along the understanding-system dimensions of Moore and Newell are suggested by the knowledge analysis. A (very tentative) figure of 97% for Studnt's degree of assimilation is based on taking the successful encoding of a KS into a P as a unit of assimilation. The kinds of problem Studnt could do, based on its present knowledge and on the knowledge required to extend its performance to other classes of problems, gives an estimate of 57.5% for Studnt's degree of accommodation (this is based on crude sampling but points out how the knowledge analysis approaches the question). The present approach suggests a way that depth of understanding and error might be handled using KSs as units contributing to a particular solution, but at present nothing more precise can be said. The figures given above are not to be taken as precise measures, but rather as indicative of the potential fruitfulness of the overall approach.

We started out this study of STUDENT by asking questions related to its intelligence and understanding, from the viewpoint of an analysis of AI programs. What has developed is an elaboration of the use of models and particularly of the concept of problem space. Intelligence is seen as knowledge in a problem space, in the form of plans, that guides the application of other knowledge as a solution is sought. The plans in Studnt have been explicitly pointed out, and a better understanding of Studnt's use of the match method has been reached. What Studnt understands is made manifest in the 218 KSs, along with our abstract characterizations of them. Further work to verify and extend the analysis will tell us how applicable it is. The details must be verified by deepening the formalization and by automation. The breadth of scope of the model will be realized from studies at a level comparable to the present study, on a wide variety of AI programs.

## D.1. Acknowledgements

The initial motivation for undertaking an analysis of AI programs and for the use of a PS for STUDENT as a means to that end was provided by Allen Newell. He also made many useful comments on the work as it developed. Any ideas that seem to lack adequate scientific support or any failures to consider the right issues relating to the problems discussed remain the author's responsibility.

## E. References

Becker, J. D., 1973. "A model for the encoding of experiential information", in Schank, R. C. and Colby, K. M., Eds., *Computer Models of Thought and Language*, San Francisco, Ca: W. H. Freeman and Company. Chapter 10.

Bobrow, D. G., 1964a. "Natural language input for a computer problem solving system", MIT Ph.D. Thesis, report MAC TR-1. Reprinted in Minsky, M., Ed., *Semantic Information Processing*, pp. 133-215. Cambridge, Ma: The MIT Press, 1968.

Bobrow, D. G., 1964b. "A question-answering system for high-school algebra word problems", *Proc. of AFIPS Fall Joint Computer Conference, 1964*, pp. 591-614.

Bobrow, D. G. and Raphael, B. R., 1973. "New programming languages for AI research", Tutorial paper for Third International Joint Conference on Artificial Intelligence.

Buchanan, J. R., 1974. "A study in automatic programming", Pittsburgh, Pa: Carnegie-Mellon University, Department of Computer Science.

Charniak, E., 1972. "Toward a model of children's story comprehension", TR-266. Cambridge, MA: MIT AI Lab. Ph. D. Thesis.

Feigenbaum, E. A., 1963. "The simulation of verbal learning behavior", in Feigenbaum, E. A. and Feldman, J., Eds., *Computers and Thought*, pp. 297-309. New York, NY: McGraw-Hill.

Hayes, J. R. and Simon, H. A., 1973. "Understanding Written Problem Instructions", Complex Information Processing Working Paper 236. Pittsburgh, Pa: Carnegie-Mellon University, Department of Psychology. This also appears in the same volume as Moore and Newell, 1973, below.

Hayes-Roth, F. and Mostow, D. J., 1975. "An automatically compilable recognition network for structured patterns", Pittsburgh, Pa: Carnegie-Mellon University, Department of Computer Science.

McCarthy, J., 1958. "Programs with common sense", *Mechanisation of Thought Processes*, Vol. 1, pp. 77-84. Reprinted in Minsky, 1968 (see Bobrow, 1964a, above), pp. 403-418.

Moore, J. and Newell, A., 1973. "How can MERLIN understand?", in Gregg, L., Ed., *Knowledge and Cognition*, pp. 201-252. Potomac, Md: Lawrence Erlbaum Associates.

Newell, A., 1969. "Heuristic programming: ill-structured problems", in Aronofsky, J. S., Ed., *Progress in Operations Research*, Vol. Vol. III, New York, NY: John Wiley. Chapter 10.

Newell, A., 1973. "Production systems : models of control structures", in Chase, W. C., Ed., *Visual Information Processing*, pp. 463-526. New York, NY: Academic Press.

Newell, A. and Simon, H. A., 1963. "GPS, a program that simulates human thought", in Feigenbaum, E. A. and Feldman, J., Eds., *Computers and Thought*, pp. 279-293. New York, NY: McGraw-Hill.

Newell, A. and Simon, H. A., 1972. *Human Problem Solving*, Englewood Cliffs, N.J: Prentice-Hall.

Paige, J. M. and Simon, H. A., 1966. "Cognitive processes in solving algebra word problems", in Kleinmutz, B., Ed., *Problem Solving: Research, Method, and Theory*, New York, NY: John Wiley. Chapter 3.

Rosenbach, J. B., Whitman, E. A., Meserve, B. E. and Whitman, P. M., 1958. *College Algebra*, Fourth edition. pp. 164-167. Boston, Ma: Ginn and Co.

Rychener, M. D., 1976. "Production systems as a programming language for artificial intelligence applications", Pittsburgh, Pa: Carnegie-Mellon University, Department of Computer Science. In preparation.

Shortliffe, E. H., 1974. "MYCIN: A rule-based computer program for advising physicians regarding antimicrobial therapy selection", AIM 251. Stanford, Ca 94305: Stanford AI Laboratory. Ph.D. Thesis.

Sussman, G. J., 1973. "A computational model of skill acquisition", AI TR-297. Cambridge, Ma: MIT Artificial Intelligence Laboratory.

Waterman, D. A., 1974. "Adaptive production systems", Complex Information Processing Working Paper 285. Pittsburgh, Pa: Carnegie-Mellon University, Department of Psychology.

Waterman, D. A. and Newell, A., 1973. "PAS-II, an interactive task-free version of an automated protocol analysis system", *Proc. Third International Joint Conference on Artificial Intelligence*, pp. 431-445.

Winograd, T., 1972. *Understanding Natural Language*, New York, NY: Academic Press. Book form of Ph. D. Thesis.

## Appendix A. Short Summary of Psnlst Features

### A.1. System architecture and production format of Psnlst

A production system (PS) is a set of conditional rules, productions (Ps), that represent changes to a symbolic model of a situation along with conditions under which those changes are to be made. A production system architecture (PSA) provides: a Working Memory (WM), which contains symbol structures representing the dynamic state of the situation being modelled; a Production Memory (PM) which contains the Ps; a particular control mechanism known as the recognize-act cycle, by which Ps are repeatedly executed or fired - a P that is recognized to have its condition satisfied with respect to WM contents is fired by having its actions performed, whereupon the cycle is repeated using the new contents of WM (WM is updated by the actions of the P that is fired); and a set of conventions or ordering principles by which a single rule may be selected from the set of rules that are recognized to be satisfied by the contents of WM during any recognize-act cycle.

The Psnlst (PS analyst) is a PSA, as follows. WM is an unordered set of data items called instances. Each instance is an ordered list of two or more elements, where the first element is a member of a set of constant atoms called predicates, and where succeeding elements are either atoms or list structures - list structures however are opaque, their internal structure not being accessible to the recognition mechanism of the PSA. Instances are considered to be grouped together in the WM according to their predicates. PM is an unordered set of Ps, each consisting of a left-hand-side or LHS (the condition part) and a right-hand-side or RHS (the action part). The form of LHSs and RHSs will be discussed below. The recognize-act cycle consists of a match of the LHS to WM, resulting in bindings for variables contained in elements of the LHS. A firing then uses those bindings to create WM instances according to the elements of the RHS. Two features of the match are unusual. First, all possible matches are found, and a firing occurs immediately for each match. That is, within a single recognize-act cycle, many firings of the same producti n may occur. Second, a match must include at least one data instance that is new with respect to the P that is matched, where new is defined as having entered WM after the previous firing of the P. The action part of a recognize-act cycle consists of adding or deleting WM instances, and of optionally making changes to PM using ADDPROD and other special operators explained below.

The way Psnlst orders satisfied Ps to select one for firing (this is the fourth PSA component) is by ordering events that occur during the action part of the recognize-act cycle. This is done by using a stack memory that records, for each WM change, the set of Ps that might become satisfied as a result of the change. The stack memory is called :SMPX, stack memory for production examinations. More recent WM changes are stacked on top of older ones, so that Ps satisfied by more recent changes are guaranteed to fire, if satisfied, before Ps using older changes. The order of recency of changes with a P firing are determined by the order of conjuncts within the P's RHS. This ordering principle leaves two selection orders unspecified: if more than one P using the same WM change is satisfied, one is arbitrarily chosen to fire and the other is pushed down in :SMPX by the changes made by the selected P; if a P fires more than once in a recognize-act cycle (more than one match is found for the P), the firings are done in an arbitrary order. With respect to the former arbitrary choice, if one P is to be selected before another one that uses the same WM change, the LHSs of the two Ps must explicitly be mutually exclusive. That is, it is the user's responsibility to distinguish between don't-care and necessarily-ordered situations. Given the :SMPX mechanism for ordering P firings, the recognize-act cycle can be summarized as follows: a change occurs to WM, resulting in :SMPX entries; starting from the top of :SMPX, Ps are matched until a P condition is found to be satisfied; the actions of the satisfied P are executed, resulting in stacking up new entries in :SMPX; and so on.

The following is a Psnlst production that appears in a PS that models a hungry monkey in a room with some bananas, as the monkey recognizes its hunger and tries to reach for the bananas.

```
H1; "HUNGRY" :: HUNGRY(M) & ISMONKEY(M) & ISBANANAS(B) & LOC(B,X,Y,H)
        => GOTO(M,X,Y) & REACHFOR(M,B);
```

The name of the P is H1, its comment is "HUNGRY", and the remainder of the P gives the LHS and the RHS, separated by "=>". The LHS is a conjunction of templates for WM elements; each template is a predicate followed by a list of variables. When a match succeeds, each variable is bound to a specific token from the WM instance corresponding to the template. H1 would match a situation in which the instances (ISMONKEY MNK-1), (HUNGRY MNK-1), (ISBANANAS BAN-1), and (LOC BAN-1 I-1 J-3 K-2) are present, to produce two new instances, (GOTO MNK-1 I-1 J-3) and (REACHFOR MNK-1 BAN-1), assuming, say, that the (HUNGRY MNK-1) instance is a new one. M is bound to MNK-1, B to BAN-1, X to I-1, and so on. MNK-1 is a token for the monkey, BAN-1 for the bananas in the room, I-1 for a spatial location along the X coordinate axis, and so on. The GOTO and REACHFOR instances become instigators of further action, if Ps to model the corresponding real actions exist and if other conditions in the model are appropriate.

## A.2. Features of Psnlst programs

The notation for Ps in Psnlst is a subset of the Mlisp language, or rather a special interpretation of Mlisp expressions (see Mlisp, by D. C. Smith, a Stanford AI Lab report, available at CMU). A PS consists of one or more modules, each of which is represented as an Mlisp EXPR consisting of a BEGIN ... END block. Each module consists of optional declarations, followed by a list of labelled Ps. A P is simply a disjunction of an optional comment string and two conjunctions, the first conjunction being the LHS, the second, the RHS. A special function is used to translate these conventions into the format used internally by Psnlst.

The following presents novel syntactic features that are encountered in reading Psnlst programs:

% - the Mlisp comment character; text between %'s is ignored.
' - used to quote Lisp S-expressions
" - string constant delimiter (for instance, Psnlst comments)
; - a semicolon is used after a P name and to separate Ps
=> - this symbol separates LHSs of Ps from RHSs

:: - used to separate Psnlst comment string from associated LHS (is DEFINE'd to be OR)

? - Mlisp character-quote character; must be used for characters that have special Mlisp meanings. For instance, V?-1 is an identifier, not "V minus 1".

& - AND

<> - Mlisp syntax for (LIST ... ), the Lisp list-building function

@ - Mlisp syntax for Lisp APPEND function, for joining two lists

Summary of notation for Ps:

name ; "comment" :: LHS => RHS ;

The following comments explain other special features of Psnlst programs, but only to the extent necessary for easier reading of the programs. Examples of these features are to be found by the reader in specific PSs.

Macros: certain things that look like predicates are really macros, expanding into a sequence of predicates with arguments; these are usually expanded at load time, by user-defined Lisp programs.

NOT specifies "absence of" when it precedes LHS conjuncts; it denotes deletion when it precedes RHS conjuncts; in LHSs it may also precede a nested conjunction, NOT( ... ), in which case the conjunction is matched as if it were an LHS, and if it succeeds the LHS match fails; these negated conjunctions may be nested, that is, they may contain nested conjunctions (see also EXISTS, below).

NEGATE is a built-in macro that specifies which of the LHS conjuncts are to be negated in the RHS, by number, or by using ALL; if negative integers follow ALL as an argument, it means "ALL but" the instances specified by the negative integers; for instance, NEGATE(3) would stand for NOT ISBANANAS(B), in the above example.

SATISFIES, SATISFIES2, SATISFIES3 are special predicates for testing values of variables during the match, using Lisp predicates; the numbers 2 and 3 are the number of variable arguments (SATISFIES takes one).

VEQ(x,y) is equivalent to SATISFIES2(x,y,x EQ y), ie equality.

VNEQ(x,y) is equivalent to SATISFIES2(x,y,x NEQ y), ie, inequality.

Conjuncts in RHSs may use arbitrary expressions as arguments, to be EVAL'd as Lisp expressions during the P firing process. (Mlisp includes Algol-like arithmetic expressions.)

NONFLUENT(p) declares p to be a non-fluent, that is, an insertion of an instance of predicate p into the Working Memory does not cause any Ps to be matched for possible firings keyed to that insertion. In other words, no entry is made to :SMPX for that change.

REQUIRE(a,b,c,...) declares that a,b,c,... are required modules of the PS whose main module contains the declaration.

PSMACRO(f1,f2,...) declares files to be read to define user macros.

DCMD(f1,f2,...) declares files to be read as command (CMD) files.

EXISTS in an RHS causes creation of new objects whose names are extensions of the arguments of the EXISTS; those objects are then used in the remainder of the RHS to form instances.

EXISTS in an LHS must be in a nested expression of the form NOT( ... ); its function then is to locally declare its arguments as variables, causing them to be initialized to NIL for the match that follows, within the ( ... ).

DELAYEXPND(x) where x is some macro call: this specifies that the macro is not to be expanded when the P is inserted, but during the actual firing of the P; this is only used when the predicates of the RHS depend on values not known until run time; it can not appear in lhs's.

ADDPROD(prod,prec,comnt,lhslist,rhslist): primitive for adding a P (named prod) with comment comnt; lhslist and rhslist are lists representing new LHS and RHS; the prec argument is either a P name, indicating that prod is to be placed after it, or is taken to be the name of a new module of which prod is the first P; ADDPROD causes assertion of (ADDPRODP prod).

REPPROD(prod,comnt,lhslist,rhslist): replace comment, LHS, and RHS of prod as indicated; asserts REPPRODP(prod).

REPLHS(prod,lhslist): replace LHS of prod as indicated; asserts REPLHSP(prod).

REPRHS(prod,rhslist): replace RHS; asserts REPRHSP(prod).

REPCOMNT(prod,comnt): replace comment string; asserts REPCOMNTP(prod).

## A.3. Features of the trace output

TOP LEVEL ASSERT - the initial starting assertion, typed by user.

! - a P fired

number following ! - the firing was the number'th

P-name followed by '-' then number - the number'th firing of the P

"string" - the comment string associated with the P

USING ... - instances from the Working Memory used in matching the LHS

(xxx . yyy) ... - assignment that was made for the match: xxx was assigned the value yyy, etc.

INSERTING ... - the insertions and deletions made by the RHS

( :SMPX .... number ) - a display of :SMPX after firing; number is length of :SMPX; each entry is enclosed in []'s

EXAMINING ... - gives the name of the P and the key insertions causing the examination

/TRY - means that a non-fast-fail examination is being done; fast-fail is a quick check on whether any positive predicate has no instances, before the full-fledged match is tried (formerly /NFF)

WARNING ... - appears when an instance is inserted or deleted but was already present or absent, respectively

++ - appears for a warning for an instance insertion

+- - appears for a warning for an instance deletion

If the RHS included ADDPROD, REPPROD, REPCOMNT, REPLHS, or REPRHS, a message is printed before the INSERTING line.

PSBREAK comment AT ... - a break in execution; user interactions consist of commands in ()'s; the system responds with output dependent on the command, or with "ok"; (OK) is typed by the user to resume execution.

The above appear on a full :DVERBOS=4 or :TVERBOS = 4 trace; the following are modifications for lesser traces:

the P-firing message is all on one line
most of the EXAMINING message disappears; only the P name remains; if /TRY occurred, only the / appears (in case of verbosity 1, not even P names appear)
most of the WARNING message disappears - only the *'s remain
the USING and INSERTING lines disappear
the messages from ADDPROD et al drop out
break messages, commands, and possibly their outputs disappear

After execution, typically a DUMP occurs ( delimited by "DUMP"), followed by the output of PERFEVAL:

Run time for the present RUN invocation
A small table of figures:
EXAM is the number of examinations of Ps
TRY is the number of non-fast-fail (/TRY) examinations
FIRE is the number of P firings
WMACT is database (Working Memory) actions: insertions + deletions
E/F, E/T, T/F give ratios of the first three
the line following the numbers gives an average time figure for each of the relevant numbers in the preceding line (divides total run time by each of the numbers)
Detail on Working Memory changes; "NEW OBJECTS" are those created by EXISTS
Maximum length attained by :SMPX
CORE gives current available LISP core, plus amount used in current run
:ACTS - a list of the major actions in the current core-image
TRACE - a list of Ps that fired, in the order that they fired
FIRED x OUT OF ... - gives number of distinct Ps that fired

# STUDNT APPENDICES

Appendix B. THE STUDENT PROGRAM

```
ORDER OF GROUPS OF PRODUCTIONS: (S T D P M C R V F A B I X)

BEGIN          % THIS COMBINES FILES STUDNT & STUDS %

EXPR STUDNT(); % PSNLST IMPLEMENTATION OF STUDENT % BEGIN

%          EXPLANATION OF MACROS :
    STRINGEQ((AA BB CC),LL,RR)
          -> LEFTOF(LL,V-1) & EQAA(V-1) & LEFTOF(V-1,V-2) & EQBB(V-2)
                & LEFTOF(V-2,V-3) & EQCC(V-3) & LEFTOF(V-3,RR)
    WORDINS(VV,'WW)
          -> EQWW(VV) & WORDEQ(VV,'WW)
    STRINGINS((AA BB CC),LL,RR)
          -> EXISTS(A1,B2,C3) & LEFTOF(LL,A1) & WORDINS(A1,'AA)
                & LEFTOF(A1,B2) & WORDINS(B2,'BB)
                & LEFTOF(B2,C3) & WORDINS(C3,'CC) & LEFTOF(C3,RR)
    INITPROB(XX,(AA BB CC))
          -> EXISTS(S0,SE) & ASCAN(XX) & PROBLEM(XX) & TGSCANFIN(S0)
                & STRINGINS((AA BB CC),S0,SE)
                & STRLENGTH(XX,LENGTH('(AA BB CC)))
                & ENDMARK(S0) & ENDMARK(SE)
%

NONFLUENT(LEFTOF WORDEQ);

REQUIRE(STUT,STUD,STUP,STUM,STUCR,STUV,STUF,STUAB,STUI);
```

```
% PAGE 2 - INITIAL LEFT-RIGHT SCAN WITH PRECEDENCE CHECK %

S10: "INIT SCAN" = TGSCANFIN(X) & ENDMARK(X) & LEFTOF(X,Y)
      -> EXISTS(C) & TFSCAN(Y) & TFSCANFIN(Y) & CHUNKENDL(Y,C) & ISSCANCHUNK(C)
            & CHUNKLEN(1) & MXCPRIOR(0) & HIGHPREC(C,0,X) & NEGATE(1):
S13: "TF SCAN" = TGSCANFIN2(X) & LEFTOF(X,Y) & NOT ISDELIM(X) & ISSCANCHUNK(C)
            & CHUNKLEN(L)
      -> TFSCAN(Y) & TFSCANFIN(Y) & INCHUNK(X,C) & CHUNKLEN(L+1)
            & NEGATE(1,5) & NOT TGSCAN(X):
S15: "TF SCAN" = TGSCANFIN2(X) & LEFTOF(X,Y) & ISSCANFV(C)
      -> TFSCAN(Y) & TFSCANFIN(Y) & INCHUNK(X,C) & NEGATE(1) & NOT TGSCAN(X)
            & NOT FVSCAN(X):
S16: "TF AGEPROB" = TFSCANFIN(X) & AGEPROB(P)
      -> TFASCAN(X) & TFASCANFIN(X) & NEGATE(1) & NOT TFSCAN(X):
S17: "TG SCAN" = TFSCANFIN(X) & NOT(EXISTS(P) & AGEPROB(P))
      -> TGSCAN(X) & TGSCANFIN(X) & NEGATE(1) & NOT TFSCAN(X):
S18: "TG SCAN" = TFASCANFIN(X)
      -> TGSCAN(X) & TGSCANFIN(X) & NEGATE(1) & NOT TFASCAN(X):

S20: "PREC SCAN" = TGSCANFIN(X) & HASPREC(X,N) & ISSCANCHUNK(C) & HIGHPREC(C,M,Y)
            & SATISFIES2(N,M,'GREAT M)
      -> TGSCANFIN2(X) & HIGHPREC(C,N,X) & NEGATE(1,4):
S25: "NO PREC" = TGSCANFIN(X) & ISSCANCHUNK(C) & NOT(EXISTS(N) & HASPREC(X,N))
      -> TGSCANFIN2(X) & NEGATE(1):
S30: "PREC LOWER" = TGSCANFIN(X) & ISSCANCHUNK(C) & HASPREC(X,N) & HIGHPREC(C,M,Y)
            & NOT SATISFIES2(M,N,'GREAT M)
      -> TGSCANFIN2(X) & NEGATE(1):
S35: "SCAN FV" = TGSCANFIN(X) & ISSCANFV(C)
      -> FVSCAN(X) & TGSCANFIN2(X) & NEGATE(1):

S40: "SCAN CHUNK" = TGSCANFIN2(X) & LEFTOF(X,V) & ISDELIM(X) & LOPERIOD(K)
            & LEFTOF(W,K) & ISSCANCHUNK(CC) & MXCPRIOR(N)
      -> PRECSCAND(CC) & TFSCAN(Y) & TFSCANFIN(Y) & ISCHUNK(CC) & CHUNKENDR(W,CC)
            & LABELLED(CC,'LTOP) & EXISTS(C) & ISSCANCHUNK(C) & CHUNKENDL(Y,C)
            & HIGHPREC(C,0,Y) & HASCPRIOR(CC,N+1) & MXCPRIOR(N+1)
            & EXISTS(CHUM) & INCHUNK(X,CHUM) & NEGATE(1,6,7)
            & CHUNKLEN(1) & NOT CHUNKLEN(0):

S60: "FV SCAN" = ISQWORD(X,X) & CHUNKENDL(X,C) & ISSCANCHUNK(C) & HIGHPREC(C,N,I)
      -> NEWFV(C) & ISFV(C) & ISSCANFV(C) & NEGATE(3,4):
S65: "FV END" = FVSCANFMD(X,DC) & LEFTOF(X,Y) & NOT ENDMARK(Y) & MXCPRIOR(N)
      -> EXISTS(C) & TFSCAN(Y) & TFSCANFIN(Y) & ISSCANCHUNK(C) & CHUNKENDL(C,Y)
            & HIGHPREC(C,0,Y) & MXCPRIOR(N+1) & HASCPRIOR(C,N+1)
            & NEGATE(1,4) & NOT ISSCANFV(CC) & NOT TGSCANFIN2(X):
S70: "RIGHT END" = FVSCANFMD(X,C) & LEFTOF(X,Y) & PROBLEM(P) & ENDMARK(V)
      -> ANSWERBUILD(P) & NEGATE(1):
```

```
END:
```

```
% PAGE 3 - TRANSFORMATIONS %

EXPR STUT(); BEGIN PSMACRO(STUDMM);

T1: "HOW OLD->WHAT" = TFSCAN(X) & EQHOW(X) & LEFTOF(X,V) & EQOLD(Y) & LEFTOF(Y,Z)
      -> MODLEN(-1) & EQWHAT(X) & WORDEQ(X,'WHAT) & NOT WORDEQ(X,'HOW)
            & LEFTOF(X,Z) & NEGATE(ALL):
T2: "IS EQUAL TO->IS" = TFSCAN(X) & EQIS(X) & STRINGEQ('(EQUAL TO),X,Y)
      -> MODLEN(-2) & LEFTOF(X,Y) & NEGATE(ALL,-2):
T2E: "EQUALS->IS" = TFSCAN(X) & EQEQUALS(X)
      -> EQIS(X) & WORDEQ(X,'IS) & NOT WORDEQ(X,'EQUALS) & NEGATE(ALL):
T3: "YEARS YOUNGER THAN->LESSTHAN" = TFSCAN(V?-1) & STRINGEQ('(YEARS YOUNGER THAN),X,Y)
      -> MODLEN(-2) & EQLESSTHAN(V?-1) & WORDEQ(V?-1,'LESSTHAN)
            & NOT WORDEQ(V?-1,'YEARS) & LEFTOF(Y?-1,Y) & NEGATE(ALL,-2):
T4: "YEARS OLDER THAN->PLUS" = TFSCAN(V?-1) & STRINGEQ('(YEARS OLDER THAN),X,Y)
      -> MODLEN(-2) & EQPLUS(V?-1) & WORDEQ(V?-1,'PLUS)
            & NOT WORDEQ(V?-1,'YEARS) & LEFTOF(V?-1,Y) & NEGATE(ALL,-2):
T5: "PERCENT LESS THAN CONV" = TFSCAN(V?-1) & STRINGEQ('(PER CENT LESS THAN),X,Y)
            & WORDEQ(X,XW) & SATISFIES(XW,NUMBERP XW)
      -> MODLEN(-3) & WORDEQ(X,(100.0-XW)/100.0) & EQTIMES(V?-1)
            & WORDEQ(V?-1,'TIMES) & NOT WORDEQ(V?-1,'PER)
            & LEFTOF(V?-1,Y) & NEGATE(ALL,-2):
      % # PER CENT LESS THAN Y -> (100-#)/100.0 TIMES Y %
T6: "LESS THAN->LESSTHAN" = TFSCAN(V?-1) & STRINGEQ('(LESS THAN),X,Y)
      -> MODLEN(-1) & EQLESSTHAN(V?-1) & WORDEQ(V?-1,'LESSTHAN)
            & NOT WORDEQ(V?-1,'LESS) & LEFTOF(V?-1,V) & NEGATE(ALL,-2):
T7: "THESE->THE" = TFSCAN(T1) & EQTHESE(T1)
      -> EQTHE(T1) & WORDEQ(T1,'THE) & NOT WORDEQ(T1,'THESE) & NEGATE(ALL):
T8: "MORE THAN->PLUS" = TFSCAN(V?-1) & STRINGEQ('(MORE THAN),X,Y)
      -> MODLEN(-1) & EQPLUS(V?-1) & WORDEQ(V?-1,'PLUS)
            & NOT WORDEQ(V?-1,'MORE) & LEFTOF(V?-1,Y) & NEGATE(ALL,-2):
T9: "SPLIT TWO NUMBERS" = TFSCAN(V?-1) & STRINGEQ('(FIRST TWO NUMBERS),X,Y)
      -> MODLEN(3) & STRINGINS('(NUMBER AND THE SECOND NUMBER),V?-1,Y)
            & NEGATE(ALL,-2,-3):
T9F: "SPLIT TWO NUMBERS" = TFSCAN(V?-1) & STRINGEQ('(TWO NUMBERS),X,Y)
      -> MODLEN(4) & EQFIRST(V?-1) & WORDEQ(V?-1,'FIRST)
            & STRINGINS('(NUMBER AND THE SECOND NUMBER),V?-1,Y)
            & NEGATE(ALL,-2) & NOT WORDEQ(V?-1,'TWO):
T10: "SPLIT THREE NUMBERS" = TFSCAN(V?-1) & STRINGEQ('(THREE NUMBERS),X,Y)
      -> MODLEN(9) & STRINGINS('(FIRST NUMBER AND THE SECOND NUMBER
            AND THE THIRD NUMBER),V?-1,Y) & WORDEQ(V?-1,'THE)
            & EQTHE(V?-1) & NOT WORDEQ(V?-1,'THREE) & NEGATE(ALL,-2):
T11: "HALF->0.5" = TFSCAN(V?-1) & STRINGEQ('(ONE HALF),X,V)
      -> MODLEN(-1) & EQ0.5(V?-1) & LEFTOF(V?-1,Y)
            & WORDEQ(V?-1,'0.5) & NOT WORDEQ(V?-1,'ONE) & NEGATE(ALL,-2):
T12: "TWICE->TWO TIMES" = TFSCAN(V?-1) & STRINGEQ('(TWICE),X,Y)
      -> MODLEN(1) & EQ2(V?-1) & WORDEQ(V?-1,'2)
            & NOT WORDEQ(V?-1,'TWICE) & STRINGINS('(TIMES),V?-1,Y)
            & NEGATE(ALL,-2):
T13: "$ SIGN" = TFSCAN(X) & EQ?$(X) & LEFTOF(X,Y) & LEFTOF(V,X)
            & WORDEQ(Y,WW) & SATISFIES(WW,'(NUMBERP WW)) & LEFTOF(Y,Z)
      -> TFOUT(X,Y) & EQDOLLARS(X) & WORDEQ(X,'DOLLARS) & NOT WORDEQ(X,'?$)
            & LEFTOF(V,V) & LEFTOF(Y,X) & LEFTOF(X,Z) & NEGATE(ALL,-5):
      % V $ # Z -> V # DOLLARS Z %
T14: "CONSEC TO->PLUS" = TFSCAN(V?-1) & STRINGEQ('(CONSECUTIVE TO),X,Y)
      -> EQ1(V?-1) & WORDEQ(V?-1,'1) & NOT WORDEQ(V?-1,'CONSECUTIVE)
            & STRINGINS('(PLUS),V?-1,V) & NEGATE(ALL,-2):
T15: "LARGER THAN->PLUS" = TFSCAN(V?-1) & STRINGEQ('(LARGER THAN),X,Y)
      -> MODLEN(-1) & WORDEQ(V?-1,'PLUS) & NOT WORDEQ(V?-1,'LARGER)
            & EQPLUS(V?-1) & LEFTOF(V?-1,Y) & NEGATE(ALL,-2):
T16: "PER CENT CONV" = TFSCAN(V?-1) & STRINGEQ('(PER CENT),X,Y)
            & WORDEQ(X,XW) & SATISFIES(XW,NUMBERP XW)
            & NOT (EXISTS(V?-3,V?-4,Z) & STRINGEQ('(LESS THAN),V?-2,Z) )
      -> MODLEN(-2) & TFOUT(V?-1,Y) & WORDEQ(X,XW/100.0) & LEFTOF(X,Y)
            & NEGATE(ALL):
      % # PER CENT Y -> #/100.0 Y %
T17: "HOW MANY->HOWM" = TFSCAN(V?-1) & STRINGEQ('(HOW MANY),X,Y)
      -> MODLEN(-1) & EQHOWM(V?-1) & WORDEQ(V?-1,'HOWM) & NOT WORDEQ(V?-1,'HOW)
            & LEFTOF(V?-1,Y) & NEGATE(ALL,-2):
T18: "THE SQUARE OF->SQUARE" = TFSCAN(V?-1) & STRINGEQ('(THE SQUARE OF),X,Y)
      -> MODLEN(-2) & EQSQUARE(V?-1) & WORDEQ(V?-1,'SQUARE)
            & NOT WORDEQ(V?-1,'THE) & LEFTOF(V?-1,V) & NEGATE(ALL,-2):
T19: "MULTIPLIED->TIMES" = TFSCAN(V?-1) & STRINGEQ('(MULTIPLIED BY),X,Y)
            & NOT EQIS(X)
      -> MODLEN(-1) & WORDEQ(V?-1,'TIMES) & EQTIMES(V?-1)
```

```
            & NOT WORDEQ(V2-1,MULTIPLIED) & LEFTOF(V2-1,Y) & NEGATE(ALL,-2);
T20: "DIVIDED-QUOTIENT" = TESCAN(V2-1) & STRINGEQ('(DIVIDED BY),X,Y)
            & NOT EQIS(X)
      -> MODLEN(-1) & WORDEQ(V2-1,QUOTIENT) & EQQUOTIENT(V2-1)
            & NOT WORDEQ(V2-1,DIVIDED) & LEFTOF(V2-1,Y) & NEGATE(ALL,-2);
T21: "SUM OF" = TESCAN(V2-1) & STRINGEQ('(THE SUM OF),W,Y) & ISSCANCHUNK(C)
      -> MODLEN(-3) & LEFTOF(W,Y) & TFOUT(V2-1,Y) & TANDSUM(C) & NEGATE(ALL,-9);
T22: "AND-+S" = TESCAN(X) & EQAND(X) & TANDSUM(C) & ISSCANCHUNK(C)
            & NOT TANDDIFF(C)
      -> EQPLUSS(X) & WORDEQ(X,'PLUSS) & NOT WORDEQ(X,'AND) & NEGATE(12);
T23: "DIFF BETW" = TESCAN(V2-1) & STRINGEQ('(THE DIFFERENCE BETWEEN),W,Y)
            & ISSCANCHUNK(C)
      -> MODLEN(-3) & LEFTOF(W,Y) & TFOUT(V2-1,Y) & TANDDIFF(C) & NEGATE(ALL,-9);
T24: "AND-+S" = TESCAN(X) & EQAND(X) & TANDDIFF(C) & ISSCANCHUNK(C)
      -> EQMINUSS(X) & WORDEQ(X,'MINUSS) & NOT WORDEQ(X,'AND) & NEGATE(12,3);
T25: "&-," = TESCAN(X) & EQ?(X) & LEFTOF(X,Y) & EQAND(Y) & LEFTOF(Y,Z)
      -> MODLEN(-1) & EQPERIOD(X) & LEFTOF(X,Z) & NEGATE(ALL);
T26: ",." = TESCAN(X) & EQ?(X) & LEFTOF(X,Y) & NOT EQAND(Y) & TFDELETED(P)
      -> EQPERIOD(X) & NEGATE(12);
T26D: ",-DEL" = TESCAN(X) & EQ?(X) & LEFTOF(X,Y) & NOT EQAND(Y) & LEFTOF(W,X)
            & NOT(EXISTS(P) & TFDELETED(P))
      -> MODLEN(-1) & TFOUT(X,Y) & LEFTOF(W,Y) & NEGATE(ALL,-2);
T27: "?-QMARK" = TESCAN(X) & EQ?(X) -> EQQMARK(X) & NEGATE(ALL);
T28: ".-PERIOD" = TESCAN(X) & EQ?(X) -> EQPERIOD(X) & NEGATE(ALL);
T29: "IF DEL" = TESCAN(X) & EQIF(X) & LEFTOF(W,X) & LEFTOF(X,Y)
      -> MODLEN(-1) & TFOUT(X,Y) & TFDELETED(X) & LEFTOF(W,Y) & NEGATE(ALL);
T30: "TOTAL #-#" = TESCAN(X) & EQTOTAL(X) & LEFTOF(W,X) & LEFTOF(X,Y)
            & EQNUMBER(Y)
      -> MODLEN(-1) & TFOUT(X,Y) & LEFTOF(W,Y) & NEGATE(ALL,-5);
T31: "EXCEEDS" = TESCAN(X) & EQEXCEEDS(X) & ISSCANCHUNK(C)
      -> EQMINUS(X) & WORDEQ(X,'MINUS) & TBYIS(C) & NEGATE(12)
            & NOT WORDEQ(X,'EXCEEDS);
T32: "BY-IS" = TESCAN(X) & EQBY(X) & TBYIS(C)
      -> EQIS(X) & WORDEQ(X,'IS) & NEGATE(ALL) & NOT WORDEQ(X,'BY);

T50: "TF OUT" = TFOUT(OW,NW) & CHUNKENDL(OW,C) & CHUNKLEN(N)
      -> TESCAN(NW) & TESCANFIN(NW) & NOT TESCANFIN(OW) & CHUNKENDL(NW,C)
            & NEGATE(ALL) & CHUNKLEN(1) & NOT TEASCANFIN(OW);
T51: "TF OUT" = TFOUT(OW,NW) & NOT (EXISTS(C) & CHUNKENDL(OW,C) )
      -> TESCAN(NW) & TESCANFIN(NW) & NOT TESCANFIN(OW) & NEGATE(ALL)
            & NOT TEASCANFIN(OW);
T52: "TF OUT LEN" = TFOUTLEN(X,YL) & CHUNKLEN(N)
      -> TFOUT(X,Y) & CHUNKLEN(N+L) & NEGATE(ALL);

END;
```

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

```
     % PAGE 4 - DICTIONARY TAGS %

EXPR STUD(): BEGIN PSMACRO(STUDNM);

D1: "LESSTHAN OP2" = TGSCAN(X) & EQLESSTHAN(X)
      -> NEWOP(X) & ISOP2(X) & NEGATE(1);
D3: "PLUS OP2" = TGSCAN(X) & EQPLUS(X) -> NEWOP(X) & ISOP2(X) & NEGATE(1);
D5: "TIMES OP1" = TGSCAN(X) & EQTIMES(X) -> NEWOP(X) & ISOP1(X) & NEGATE(1);
D7: "SQUARE OP1" = TGSCAN(X) & EQSQUARE(X) -> NEWOP(X) & ISOP1(X) & NEGATE(1);
D9: "QUOTIENT OP1" = TGSCAN(X) & EQQUOTIENT(X)
      -> NEWOP(X) & ISOP1(X) & NEGATE(1);
D11: "OF OP1" = TGSCAN(X) & EQOF(X) -> ISOP1(X) & NEGATE(1);
D13: "SQUARED OP" = TGSCAN(X) & EQSQUARED(X) -> NEWOP(X) & ISOP0(X) & NEGATE(1);
D16: "EXPONENT OP" = TGSCAN(X) & EQ?*?*(X)
      -> NEWOP(X) & ISOP0(X) & WORDIS(X,'EXPT) & NEGATE(12)
            & NOT WORDEQ(X,'?*?*);
D15: "MINUS OP" = TGSCAN(X) & EQMINUS(X) -> NEWOP(X) & ISOP0(X) & NEGATE(1);
D17: "PER OP" = TGSCAN(X) & EQPER(X)
      -> NEWOP(X) & ISOP0(X) & WORDIS(X,'QUOTIENT) & NEGATE(ALL)
            & NOT WORDEQ(X,'PER);
D18: "PLUSS OP" = TGSCAN(X) & EQPLUSS(X) -> NEWOP(X) & ISOP0(X) & NEGATE(1);
D19: "MINUSS OP" = TGSCAN(X) & EQMINUSS(X) -> NEWOP(X) & ISOP0(X) & NEGATE(1);
D21: "HAS VB" = TGSCAN(X) & EQHAS(X) -> ISVERB(X) & NEGATE(1);
D24: "GETS VB" = TGSCAN(X) & EQGETS(X) -> ISVERB(X) & NEGATE(1);
D27: "HAVE VB" = TGSCAN(X) & EQHAVE(X) -> ISVERB(X) & NEGATE(1);
D30: "WEIGHS VB" = TGSCAN(X) & EQWEIGHS(X) -> ISVERB(X) & NEGATE(1);
D41: "MARY PR" = TGSCAN(X) & EQMARY(X) -> ISPERSON(X) & NEGATE(1);
D44: "ANN PR" = TGSCAN(X) & EQANN(X) -> ISPERSON(X) & NEGATE(1);
D47: "BILL PR" = TGSCAN(X) & EQBILL(X) -> ISPERSON(X) & NEGATE(1);
D50: "FATHER PR" = TGSCAN(X) & EQFATHER(X) -> ISPERSON(X) & NEGATE(1);
D53: "UNCLE PR" = TGSCAN(X) & EQUNCLE(X) -> ISPERSON(X) & NEGATE(1);
```

```
D55: "HE PRON" = TGSCAN(X) & AGEPROB(P) & EQHE(X) -> ISPRON(X) & NEGATE(1);
D57: "HIS POSS" = TGSCAN(X) & AGEPROB(P) & EQHIS(X)
      -> ISPOSSPRON(X) & NEGATE(1);
D61: "PEOPLE PL" = TGSCAN(X) & EQPEOPLE(X) -> ISPLURAL(X,'PERSON) & NEGATE(ALL);
D63: "FEET PL" = TGSCAN(X) & EQFEET(X) -> ISPLURAL(X,'FOOT) & NEGATE(ALL);
D65: "YARDS PL" = TGSCAN(X) & EQYARDS(X) -> ISPLURAL(X,'YARD) & NEGATE(ALL);
D67: "FATHOMS PL" = TGSCAN(X) & EQFATHOMS(X) -> ISPLURAL(X,'FATHOM) & NEGATE
D67S: "FATHOM SING" = TGSCAN(X) & EQFATHOM(X) -> ISSINGULAR(X) & NEGATE(1);
D69: "INCHES PL" = TGSCAN(X) & EQINCHES(X) -> ISPLURAL(X,'INCH) & NEGATE(ALL);
D71: "SPANS PL" = TGSCAN(X) & EQSPANS(X) -> ISPLURAL(X,'SPAN) & NEGATE(ALL);
D71S: "SPAN SING" = TGSCAN(X) & EQSPAN(X) -> ISSINGULAR(X) & NEGATE(1);
D72: "MILES PL" = TGSCAN(X) & EQMILES(X) -> ISPLURAL(X,'MILE) & NEGATE(ALL);
D73: "GALLONS PL" = TGSCAN(X) & EQGALLONS(X) -> ISPLURAL(X,'GALLON) & NEGAT
D75: "HOURS PL" = TGSCAN(X) & EQHOURS(X) -> ISPLURAL(X,'HOUR) & NEGATE(ALL);
D77: "POUNDS PL" = TGSCAN(X) & EQPOUNDS(X) -> ISPLURAL(X,'POUND) & NEGATE(AL
D78: "TONS PL" = TGSCAN(X) & EQTONS(X) -> ISPLURAL(X,'TON) & NEGATE(ALL);
D79: "DOLLARS PL" = TGSCAN(X) & EQDOLLARS(X) -> ISPLURAL(X,'DOLLAR) & NEGATE(   );
D81: "WHAT QW" = TGSCAN(X) & EQWHAT(X) -> ISQWORD(X) & NEGATE(1);
D83: "FIND QW" = TGSCAN(X) & EQFIND(X) -> ISQWORD(X) & NEGATE(1);
D85: "HOWM QW" = TGSCAN(X) & EQHOWM(X) -> ISQWORD(X) & NEGATE(1);
D87: "HOW QW" = TGSCAN(X) & EQHOW(X) & LEFTOF(X,Y) & NOT EQOLD(Y)
            & NOT EQMANY(Y)
      -> ISQWORD(X) & NEGATE(1);
D91: "PERIOD DLM" = TGSCAN(X) & EQPERIOD(X)
            -> MODLENC(X) & ISDELIM(X) & NEGATE(1);
D96: "IS CHNK" = TGSCAN(X) & EQIS(X) -> MODLENC(X) & ISIS(X) & NEGATE(1);

END;
```

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

```
     % PAGE 5 - PRECEDENCE SCAN %

EXPR STUP(): BEGIN

P1: "VERB PREC" = ISVERB(X) -> HASPREC(X,9);
P2: "IS PREC" = EQIS(X) & ISIS(X) -> HASPREC(X,8);
P3: "OP2 PREC" = ISOP2(X) -> HASPREC(X,7);
P4: "SQUARE PREC" = EQSQUARE(X) & ISOP1(X) -> HASPREC(X,6);
P5: "OP1 PREC" = ISOP1(X) & NOT EQSQUARE(X) & NOT EQOF(X) -> HASPREC(X,5);
P8: "SQUARED PREC" = EQSQUARED(X) & ISOP0(X) -> HASPREC(X,2);
P9: "OP0 PREC" = ISOP0(X) & NOT EQSQUARED(X) -> HASPREC(X,1);

P10: "START PREC SCAN" = ISCHUNK(C0) & CHUNKENDL(X,C0) & HASCPRIOR(C0,M0)
            & NOT PRECSCAND(C0) & ISCHUNK(C1) & HASCPRIOR(C1,M1)
            & SATISFIES2(M0,M1,M0 ?*GREAT M1) & NOT PRECSCAND(C1)
            & NOT( EXISTS(C2,M2) & HASCPRIOR(C2,M2)
                  & SATISFIES2(M0,M2,M2 ?*GREAT M0) & NOT PRECSCAND(C2) )
            & NOT( EXISTS(C3,M3) & HASCPRIOR(C3,M3)
                  & SATISFIES3(M0,M1,M3,'(GREATERP M0 M3 M1))
                  & NOT PRECSCAND(C3) )
      -> PRECSCAN(C0,X) & HIGHPREC(C0,0,X) & ISCHUNK(C1);
P15: "START PREC SCAN" = ISCHUNK(C0) & CHUNKENDL(X,C0) & HASCPRIOR(C0,M0)
            & NOT PRECSCAND(C0)
            & NOT( EXISTS(C1,M1) & HASCPRIOR(C1,M1)
                  & SATISFIES2(M0,M1,M0 ?*GREAT M1) & NOT PRECSCAND(C1) )
            & NOT (EXISTS(C2,M2) & HASCPRIOR(C2,M2)
                  & SATISFIES2(M0,M2,M2 ?*GREAT M0) & NOT PRECSCAND(C2) )
      -> PRECSCAN(C0,X) & HIGHPREC(C0,0,X);

P20: "NEW HIGH PREC" = PRECSCAN(C,X) & HIGHPREC(C,N,Y) & HASPREC(X,M)
            & SATISFIES2(M,N,'(GREATERP M N)) & LEFTOF(X,W)
            & NOT CHUNKENDR(X,C)
      -> PRECSCAN(C,W) & HIGHPREC(C,M,X) & NEGATE(12);

P23: "PREC SCAN ON" = PRECSCAN(C,X) & HIGHPREC(C,N,Y) & HASPREC(X,M)
            & NOT SATISFIES2(M,N,'(GREATERP M N)) & LEFTOF(X,W)
            & NOT CHUNKENDR(X,C)
      -> PRECSCAN(C,W) & NEGATE(1);
P26: "PREC SCAN ON" = PRECSCAN(C,X) & NOT( EXISTS(N) & HASPREC(X,N) )
            & LEFTOF(X,W) & NOT CHUNKENDR(X,C)
      -> PRECSCAN(C,W) & NEGATE(1);

P27: "PREC SCAN DONE" = PRECSCAN(C,X) & HIGHPREC(C,N,Y) & HASPREC(X,M)
            &SATISFIES2(M,N,'(GREATERP M N)) & CHUNKENDR(X,C)
      -> HIGHPREC(C,M,X) & PRECSCAND(C) & NEGATE(12);
P28: "PREC SCAN DONE" = PRECSCAN(C,X) & HIGHPREC(C,N,Y) & HASPREC(X,M)
            & NOT SATISFIES2(M,N,'(GREATERP M N)) & CHUNKENDR(X,C)
      -> PRECSCAND(C) & NEGATE(1);
```

```
P29: "PREC SCAN DONE" = PRECSCAN(C,X) & NOT( EXISTS(N) & HASPREC(X,N) )
          & CHUNKENDR(X,C)
     -> PRECSCAND(C) & NEGATE(1);

P30: "HASVERB" = PRECSCAND(C) & HIGHPREC(C,M,X) & SATISFIES(M,'(EQ M 9))
     -> HASVERB(C,X) & NEGATE(2);
P35: "HASIS" = PRECSCAND(C) & HIGHPREC(C,M,X) & SATISFIES(M,'(EQ M 8))
     -> HASIS(C,X) & NEGATE(2);
P40: "HASOP2" = PRECSCAND(C) & HIGHPREC(C,M,X) & SATISFIES(M,'(EQ M 7))
     -> HASOP2(C,X) & NEGATE(2);
P45: "HASSQUARE" = PRECSCAND(C) & HIGHPREC(C,M,X) & SATISFIES(M,'(EQ M 6))
     -> HASSQUARE(C,X) & NEGATE(2);
P50: "HASOP1" = PRECSCAND(C) & HIGHPREC(C,M,X) & SATISFIES(M,'(EQ M 5))
     -> HASOP1(C,X) & NEGATE(2);
P65: "HASSQUARED" = PRECSCAND(C) & HIGHPREC(C,M,X) & SATISFIES(M,'(EQ M 2))
     -> HASSQUARED(C,X) & NEGATE(2);
P70: "HASOPO" = PRECSCAND(C) & HIGHPREC(C,M,X) & SATISFIES(M,'(EQ M 1))
     -> HASOPO(C,X) & NEGATE(2);
P75: "VAR FOUND" = PRECSCAND(C) & HIGHPREC(C,M,X) & SATISFIES(M,'(EQ M 0))
          & LABELLU(C,NP)
     -> ISVARCHUNK(C) & LABELF(C,NP) & NEGATE(2,0);

END;
```

. . . . . . . . . . . . . .   . . . . . . . .

```
     % PAGE 6 - MATH CONNECTIVES, VERBS, MISC. POST-DICT-TAG TRANSLS %

EXPR STUM(); BEGIN     RSMACRO(STUDNM);

M10: "CONN =" = EQIS(X) & HASIS(C,X) & LEFTOF(X,A2)
          & NOT EQMULTIPLIED(A2) & NOT EQDIVIDED(A2) & NOT EQINCREASED(A2)
     -> NEWEQN(X) & CSPLIT(C,X,X) & HASOP(C,'EQUAL) & NEGATE(2);

M20: "AS MANY AS VB" = ISVERB(V1) & INCHUNK(V1,C) & HASVERB(C,V1)
          & CHUNKENDL(A1L,C) & LEFTOF(A1R,V1) & LEFTOF(V1,A2L)
          & LEFTOF(A2R,AS1) & EQAS(AS1) & VNEQ(V1,A2R) & INCHUNK(AS1,C)
          & LEFTOF(AS1,MANY1) & LEFTOF(MANY1,A3L) & EQMANY(MANY1)
          & LEFTOF(A3R,AS2) & EQAS(AS2) & VNEQ(AS1,AS2) & ISVERB(V2)
          & INCHUNK(V2,C) & VNEQ(V1,V2) & LEFTOF(AS2,A4L) & LEFTOF(A4R,V2)
          & LEFTOF(V2,A5L) & CHUNKENDR(A5R,C)
          & LABELLU(C,NP) & HASCPRIOR(C,M) & LEFTOF(A5R,Z)
     -> EXISTS(CN) & PRENAME(A2L,C,CN) & CHUNKENDL(A2L,CN) & CHUNKENDR(V2,CN)
          & LABELLU(CN,NP) & HASCPRIOR(CN,M)
          & STRINGINS('(THE NUMBER OF),A2R,A3L) & LEFTOF(A3R,A1L)
          & STRINGINS('(IS THE NUMBER OF),V1,A5L) & LEFTOF(A5R,A4L)
          & HASPREC(10,R) & NEGATE(ALL,-2,-5,-8,-13,-15,-21)
          & NOT INCHUNK(MANY1,C) & NOT HASPREC(V1,9) & NOT HASPREC(V2,9);

     % M20: A1L...A1R-V1-A2L...A2R-AS1-MANY1-A3L...A3R-AS2-A4L...A4R-V2-A5L...A5R
          -> A2L...A2R-(THE # OF)-A3L...A3R-A1L...A1R-V1-(IS THE # OF)
               -A5L...A5R-A4L...A4R-V2 %
     % WHERE AnL ... AnR STANDS FOR THE nTH ARBITRARY PHRASE,
          Vn FOR nTH VERB, ASn FOR nTH AS, ETC. %

M30: "AS MANY AS VB" = ISVERB(V1) & INCHUNK(V1,C) & HASVERB(C,V1)
          & CHUNKENDL(A1L,C) & LEFTOF(A1R,V1)
          & LEFTOF(V1,AS1) & EQAS(AS1) & LEFTOF(AS1,MANY1)
          & LEFTOF(MANY1,A3L) & EQMANY(MANY1) & LEFTOF(A3R,AS2)
          & EQAS(AS2) & VNEQ(AS1,AS2) & INCHUNK(AS2,C) & LEFTOF(AS2,A4L)
          & ISVERB(V2) & VNEQ(V1,V2) & INCHUNK(V2,C) & LEFTOF(A4R,V2)
          & LEFTOF(V2,A5L) & CHUNKENDR(A5R,C)
          & LABELLU(C,NP) & HASCPRIOR(C,M) & LEFTOF(A5R,Z)
     -> EXISTS(CN,T1) & PRENAME(T1,C,CN) & CHUNKENDL(T1,CN) & CHUNKENDR(V2,CN)
          & LABELLU(CN,NP) & HASCPRIOR(CN,M)
          & EXISTS(DUM) & STRINGINS('(THE NUMBER OF),DUM,A3L) & LEFTOF(A3R,A1L)
          & STRINGINS('(IS THE NUMBER OF),V1,A5L) & LEFTOF(A5R,A4L)
          & HASPREC(10,R) & NEGATE(ALL,-2,-5,-7,-10,-12,-19)
          & NOT INCHUNK(AS1,C) & NOT INCHUNK(MANY1,C)
          & NOT LEFTOF(DUM,T1) & NOT HASPREC(V1,9) & NOT HASPREC(V2,9);

     % M30: A1L...A1R-V1-AS1-MANY1-A3L...A3R-AS2-A4L...A4R-V2-A5L...A5R
          -> DUM-(THE # OF)-A3L...A3R-A1L...A1R-V1-(IS THE # OF)
               -A5L...A5R-A4L...A4R-V2 %

M40: "HASVERB OLL" = HASVERB(C,V)
          & NOT( EXISTS(A,M) & EQAS(A) & EQMANY(M) & LEFTOF(A,M)
               & INCHUNK(A,C) )
          & NOT( EXISTS(N,NW) & LEFTOF(V,N) & WORDEQ(N,NW)
               & SATISFIES(NW,'NUMBERP NW) )
     -> ISCHUNK(C) & NOT PRECSCAND(C) & NOT HASPREC(V,9) & NEGATE(1);

M50: "VB WITH #" = ISVERB(V) & INCHUNK(V,C) & LEFTOF(A1R,V) & CHUNKENDL(A1L,C)
          & HASVERB(C,V) & LEFTOF(V,N) & WORDEQ(N,WN)
          & SATISFIES(WN,'(NUMBERP WN))
          & LEFTOF(N,X) & LEFTOF(X,A2L) & CHUNKENDR(A2R,C) & VNEQ(X,A2R)
          & NOT (EXISTS(A,M) & EQAS(A) & LEFTOF(A,M) & EQMANY(M)
               & INCHUNK(A,C) )
          & LABELLU(C,M,P) & HASCPRIOR(C,O)
     -> EXISTS(CN,T1) & PRENAME(T1,C,CN) & CHUNKENDL(T1,CN) & CHUNKENDR(A2R,CN)
          & LABELLU(CN,M,P) & HASCPRIOR(CN,O)
          & EXISTS(DUM) & STRINGINS('(THE NUMBER OF),DUM,X) & LEFTOF(X,A1L)
          & STRINGINS('(IS),V,N) & LEFTOF(N,A2L) & HASPREC(10,R)
          & NEGATE(1,4,5,6,9,10,10,15) & NOT HASPREC(V,9);

     % M50: A1L...A1R-V-N-X-A2L...A2R
          -> DUM-(THE-NUMBER-OF)-X-A1L...A1R-V-(IS)-N-A2L...A2R %

M55: "VB WITH #" = ISVERB(V) & INCHUNK(V,C) & LEFTOF(A1R,V) & CHUNKENDL(A1L,C)
          & HASVERB(C,V) & LEFTOF(V,N) & WORDEQ(N,WN)
          & SATISFIES(WN,'(NUMBERP WN))
          & LEFTOF(N,X) & CHUNKENDR(X,C) & LEFTOF(X,Y)
          & NOT (EXISTS(A,M) & EQAS(A) & LEFTOF(A,M) & EQMANY(M)
               & INCHUNK(A,C) )
          & LABELLU(C,M,P) & HASCPRIOR(C,O)
     -> EXISTS(CN,T1) & PRENAME(T1,C,CN) & CHUNKENDL(T1,CN) & CHUNKENDR(N,CN)
          & LABELLU(CN,M,P) & HASCPRIOR(CN,O)
          & EXISTS(DUM) & STRINGINS('(THE NUMBER OF),DUM,X) & LEFTOF(X,A1L)
          & STRINGINS('(IS),V,N) & LEFTOF(N,Y) & HASPREC(10,R)
          & NEGATE(ALL,-2,-3,-7) & NOT HASPREC(V,9);

     % M50: A1L...A1R-V-N-X -> DUM-(THE-NUMBER-OF)-X-A1L...A1R-V-(IS)-N %

M60: "PLURAL -TIMES" = ISPLURAL(X,XS) & LEFTOF(W,X) & WORDEQ(W,WW)
          & SATISFIES(WW,'(NUMBERP WW))
          & WORDEQ(X,XP) & INCHUNK(W,C) & LEFTOF(V,W) & NOT ISVERB(V)
     -> EXISTS(T1) & NEWOP(T1) & MODLEN(1) & STRINGINS('(TIMES),W,X)
          & ISOP1(T1) & INCHUNK(T1,C) & NOT LEFTOF(W,X) & WORDEQ(X,XS)
          & NEGATE(5);
M62: "SINGULAR TIMES" = ISSINGULAR(X) & LEFTOF(W,X) & EQ1(W) & INCHUNK(W,C)
          & ISSCANCHUNK(C)
     -> EXISTS(T1) & NEWOP(T1) & MODLEN(1) & STRINGINS('(TIMES),W,X)
          & ISOP1(T1) & INCHUNK(T1,C) & NEGATE(2);
M65: "OF -TIMES" = EQOF(X) & ISOP1(X) & LEFTOF(W,X) & WORDEQ(W,WW)
          & SATISFIES(WW,'(NUMBERP WW))
     -> NEWOP(X) & EQTIMES(X) & WORDEQ(X,'TIMES) & NOT WORDEQ(X,'OF)
          & NEGATE(1);
M75: "OF NOT OP" = EQOF(X) & LEFTOF(W,X) & WORDEQ(W,WW)
          & NOT SATISFIES(WW,'(NUMBERP WW)) & ISOP1(X)
     -> NOT ISOP1(X);

END;
```

. . . . . . . . . . . . . . . . . . . . .

```
     % PAGE 7 - CHUNK SPLITTING, RE-FORMING, AND RE-NAMING %

EXPR STUCR(); BEGIN

C2: "IS MULT BY" = HASIS(C,X) & LEFTOF(X,Y) & EQMULTIPLIED(Y)
          & LEFTOF(Y,Z) & EQBY(Z)
     -> NEWPREFOP('TIMES) & CSPLIT(C,X,Z) & HASOP(C,'TIMES);
C5: "IS DIV BY" = HASIS(C,X) & LEFTOF(X,Y) & EQDIVIDED(Y) & LEFTOF(Y,Z)
          & EQBY(Z)
     -> NEWPREFOP('QUOTIENT) & CSPLIT(C,X,Z) & HASOP(C,'QUOTIENT);
C8: "IS INCR BY" = HASIS(C,X) & LEFTOF(X,Y) & EQINCREASED(Y)
          & LEFTOF(Y,Z) & EQBY(Z)
     -> NEWPREFOP('PLUS) & CSPLIT(C,X,Z) & HASOP(C,'PLUS);

C10: "OP2 BRK" = HASOP2(C,X) & WORDEQ(X,XW)
     -> CSPLIT(C,X,X) & HASOP(C,XW) & NEGATE(1);
C15: "SQUARE BRK" = HASSQUARE(C,X) & NOT CHUNKENDL(X,C) & LEFTOF(W,X)
          & CHUNKENDR(R,C) & LEFTOF(X,Y) & LABELLU(C,NP)
     -> EXISTS(CL1,DUME,DUMO,DUMV) & URENAME(C,CL1,DUMV,Y,DUMO) & ISCHUNK(CL1)
          & NOT PRECSCAND(C) & ISCHUNK(C) & LABELLU(CL1,N+7,C) & LEFTOF(R,DUMO)
          & LEFTOF(DUMO,DUMV) & CHUNKENDR(DUMV,CL1) & EQ2(DUMV)
          & WORDEQ(DUMV,'2) & EQEXPT(DUMO) & WORDEQ(DUMO,'EXPT)
          & LEFTOF(W,DUME) & ISUOPDUM(DUME) & HASUOPCHUNK(DUME,CL1)
```

```
        & CHUNKENDR(DUME,C) & INCHUNK(DUME,C) & NEGATE(ALL,6);
C47: "SQUARE BRK" = HASSQUARE(C,X) & CHUNKENDL(X,C) & LEFTOF(X,Y)
        & CHUNKENDR(P,C)
    => EXISTS(DUMO,DUMV) & CSPLIT(C,DUMO,DUMO) & HASOP(C,TXPT)
        & CHUNKENDL(Y,C) & LEFTOF(P,DUMO) & LEFTOF(DUMO,DUMV)
        & INCHUNK(DUMO,C) & INCHUNK(DUMV,C) & CHUNKENDR(DUMV,C)
        & EQ2(DUMV) & WORDEQ(DUMV,'2) & EQEXPT(DUMO) & WORDEQ(DUMO,TXPT)
        & NEGATE(ALL);

C20: "U RENAME" = URENAME(CO,CU,POS,TERM,UOP) & VNEQ(POS,TERM) & LEFTOF(X,POS)
    => URENAME(CO,CU,X,TERM,UOP) & INCHUNK(POS,CU) & NOT INCHUNK(POS,CO)
        & NEGATE(1);
C22: "U RENAMED" = URENAME(CO,CU,POS,TERM,UOP) & VEQ(POS,TERM) & WORDEQ(UOP,O)
    => CSPLIT(CU,UOP,UOP) & CHUNKENDL(POS,CU) & INCHUNK(POS,CU) & HASOP(CU,O)
        & NOT INCHUNK(POS,CO) & NEGATE(1);

C25: "OP1 BRK" = HASOP1(C,X) & WORDEQ(X,XW)
        => CSPLIT(C,X,X) & HASOP(C,XW) & NEGATE(1);

C50: "SQUARED BRK" = HASSQUARED(C,X) & NOT CHUNKENDR(X,C) & CHUNKENDL(L,C)
        & LEFTOF(X,Y) & LEFTOF(W,X) & LABELU(C,N,P)
    => EXISTS(CU,DUME,DUMO,DUMV) & URENAME(C,CU,DUMV,DUMO) & ISCHUNK(CU)
        & ISCHUNK(C) & NOT PRECSCAND(C) & LABELU(CU,N+2,C)
        & LEFTOF(W,DUMO) & LEFTOF(DUMO,DUMV) & CHUNKENDR(DUMV,CU)
        & EQ2(DUMV) & WORDEQ(DUMV,'2) & EQEXPT(DUMO) & WORDEQ(DUMO,TXPT)
        & LEFTOF(DUME,Y) & ISUOPDUM(DUME) & HASUOPCHUNK(DUME,CU)
        & INCHUNK(DUME,C) & CHUNKENDL(DUME,C) & NEGATE(ALL,6);
C52: "SQUARED BRK" = HASSQUARED(C,X) & CHUNKENDR(X,C) & LEFTOF(W,X)
    => EXISTS(DUMO,DUMV) & CSPLIT(C,DUMO,DUMO) & HASOP(C,EXPT)
        & LEFTOF(W,DUMO) & LEFTOF(DUMO,DUMV) & INCHUNK(DUMO,C)
        & INCHUNK(DUMV,C) & CHUNKENDR(DUMV,C) & EQ2(DUMV) & WORDEQ(DUMV,'2)
        & EQEXPT(DUMO) & WORDEQ(DUMO,TXPT) & NEGATE(ALL);

C55: "OPO BRK" = HASOPO(C,X) & WORDEQ(X,XW)
        => CSPLIT(C,X,X) & HASOP(C,XW) & NEGATE(1);

C60: "SPLIT CHUNK" = CSPLIT(C,LOCL,LOCR) & LEFTOF(X1,LOCL) & LEFTOF(LOCR,X2)
        & LABELU(C,N,P) & MXCPRIOR(M)
    => EXISTS(CL,CR) & NEWPLOP(C) & RRENAME(X2,C,CR) & LRENAME(X1,C,CL)
        & LABELU(CL,N+1,C) & LABELU(CR,N+1,C) & HASCPRIOR(CL,M+2)
        & HASCPRIOR(CR,M+1) & MXCPRIOR(M+2) & CHUNKENDL(X2,CR)
        & CHUNKENDR(X1,CL) & NEGATE(1,2,3,5);
C70: "FINISH SEG" = LABELU(C,N,P) & LABELF(C1,M,C) & LABELF(C2,M,C)
        & HASOP(C,X) & SATISFIES(P,P NEQ 'TOP) & HASCPRIOR(C1,PR1)
        & HASCPRIOR(C2,PR2) & SATISFIES2(PR1,PR2,PR1 ?>GREAT PR2)
        & SATISFIES2(M,N,'(EQUAL (?>DIF M N) 1))
        & HASEXPR(C1,Y) & HASEXPR(C2,Z)
    => HASEXPR(C,'X,Y,Z) & LABELF(C,N,P) & NEGATE(1);
C75: "FINISH SEG -" = LABELU(C,N,P) & LABELF(C1,M,C) & LABELF(C2,M,C)
        & HASOP(C,X) & SATISFIES(X,X EQ 'EQUAL) & HASCPRIOR(C1,PR1)
        & HASCPRIOR(C2,PR2) & SATISFIES2(PR1,PR2,PR1 ?>GREAT PR2)
        & SATISFIES(M,M EQ 2) & HASEXPR(C1,Y) & HASEXPR(C2,Z)
    => NEWREFEXPR(C1) & HASEXPR(C,'X,Y,Z) & LABELF(C,N,P) & NEGATE(1);
C78: "FINISH SEG TOP REF" = LABELU(C,N,P) & LABELF(C1,M,C) & LABELF(C2,M,C)
        & SATISFIES(P,P EQ 'TOP) & HASOP(C,X) & SATISFIES(X,X NEQ 'EQUAL)
        & HASCPRIOR(C1,PR1) & HASCPRIOR(C2,PR2)
        & SATISFIES2(PR1,PR2,PR1 ?>GREAT PR2) & SATISFIES(M,M EQ 2)
        & HASEXPR(C1,Y) & HASEXPR(C2,Z)
    => NEWREFEXPR(C) & HASEXPR(C,'X,Y,Z) & LABELF(C,N,P) & NEGATE(1);

C80: "NEW REF" = NEWREFEXPR(CN) & ISREFEXPR(CO)
    => ISREFEXPR(CN) & NEGATE(ALL);
C85: "NEW REF 1" = NEWREFEXPR(CN) & NOT( EXISTS(CO) & ISREFEXPR(CO) )
    => ISREFEXPR(CN) & NEGATE(1);

C90: "FIN EQN" = HASEXPR(C,X) & LABELF(C,N,P) & SATISFIES(N,'(EQUAL N 1))
        & HASOP(C,O) & SATISFIES(O,O EQ 'EQUAL)
    => ISEQN(C,X);

R2: "CHUNK LRENAME" = LRENAME(W,OC,NC) & LEFTOF(V,W) & INCHUNK(W,OC)
        & NOT CHUNKENDL(W,OC)
    => LRENAME(V,OC,NC) & INCHUNK(W,NC) & NEGATE(1,3);
R4: "CHUNK LRENAMED" = LRENAME(W,OC,NC) & CHUNKENDL(W,OC) & INCHUNK(W,OC)
    => ISCHUNK(NC) & INCHUNK(W,NC) & CHUNKENDL(W,NC) & NEGATE(ALL);
R6: "CHUNK RRENAME" = RRENAME(W,OC,NC) & LEFTOF(W,X) & INCHUNK(W,OC)
        & NOT CHUNKENDR(W,OC) & NOT CHUNKENDR(W,NC)
    => RRENAME(X,OC,NC) & INCHUNK(W,NC) & NEGATE(1,3);
R7: "CHUNK RRENAME" = RRENAME(W,OC,NC) & NOT INCHUNK(W,OC) & LEFTOF(W,X)
        & NOT CHUNKENDR(W,OC) & NOT CHUNKENDR(W,NC)
    => RRENAME(X,OC,NC) & INCHUNK(W,NC) & NEGATE(1);
R8: "CHUNK RRENAMED" = RRENAME(W,OC,NC) & CHUNKENDR(W,OC) & INCHUNK(W,OC)
    => ISCHUNK(NC) & INCHUNK(W,NC) & CHUNKENDR(W,NC) & NEGATE(ALL);
R9: "CHUNK RRENAMED" = RRENAME(W,OC,NC) & NOT CHUNKENDR(W,OC) & CHUNKENDR(W,NC)
    => ISCHUNK(NC) & INCHUNK(W,NC) & NEGATE(1) & NOT INCHUNK(W,OC);

END;
```

. . . . . . . . . . . . . . . . . . . . .

```
  \ PAGE 8 - VAR IDENT TESTS \

EXPR STUV(): BEGIN PSMACRO(STUDNM);

V5: "IS VAR" = ISVARCHUNK(C) & NOT(EXISTS(X) & HASEXPR(C,X) ) & NUMVARCHUNKS(V)
        & NOT (EXISTS(X) & CHUNKENDL(X,C) & ISUOPDUM(X) )
    => UNTESTED(C) & CHTESTED(C,1) & NUMVARCHUNKS(V+1) & NEGATE(3);
V10: "VAR UOP" = ISVARCHUNK(C) & CHUNKENDL(X,C) & ISUOPDUM(X)
        & HASUOPCHUNK(X,CU) & HASEXPR(CU,E)
    => HASEXPR(C,Z) & NEGATE(1);

V15: "VAR THIS" = ISVARCHUNK(C) & UNTESTED(C) & EQTHIS(Y) & INCHUNK(Y,C)
        & ISREFEXPR(C2) & HASEXPR(C2,X)
    => VARCLEANUP(C) & HASEXPR(C,X) & NEGATE(2,5);
V20: "THIS FAIL" = ISVARCHUNK(C) & UNTESTED(C)
        & NOT( EXISTS(Y) & EQTHIS(Y) & INCHUNK(Y,C) )
    => THISTESTED(C) & NEGATE(2);
V21: "THIS FAIL2" = ISVARCHUNK(C) & UNTESTED(C) & EQTHIS(Y) & INCHUNK(Y,C)
        & NOT( EXISTS(C2) & ISREFEXPR(C2) )
    => THISTESTED(C) & NEGATE(2);

V23: "COUNT EQVAR" = THISTESTED(C1) & ISVARCHUNK(C2) & EQVARCHUNK(C2,C3)
    => EQVARREMD(C1) & CHTCOUNTED(C2,C1) & NEGATE(1);
V24: "NO EQVAR" = THISTESTED(C1) & NOT( EXISTS(C2,C3) & EQVARCHUNK(C2,C3) )
    => EQVARREMD(C1) & NEGATE(1);

V25: "VAR EQ TEST" = EQVARREMD(C1) & ISVARCHUNK(C2) & VNEQ(C1,C2)
        & CHUNKENDL(X,C1) & CHUNKENDL(Y,C2)
        & NOT( EXISTS(C3) & EQVARCHUNK(C2,C3) )
    => EQCHUNKTEST(C1,C2,X,Y) & VARCHCOUNT(C1,C2) & NEGATE(1);
V26: "VAR 1" = EQVARREMD(C1) & NOT( EXISTS(C2) & ISVARCHUNK(C2) & VNEQ(C1,C2) )
    => NEGATE(1);

V30: "VAR +" = EQCHUNKTEST(C1,C2,X,Y) & WORDEQ(X,XW) & WORDEQ(Y,XW)
        & LEFTOF(X,X2) & LEFTOF(Y,Y2) & NOT CHUNKENDR(X,C1)
        & NOT CHUNKENDR(Y,C2)
    => EQCHUNKTEST(C1,C2,X2,Y2) & NEGATE(1);

V31: "THE+A" = EQCHUNKTEST(C1,C2,X,Y) & EQTHE(X) & EQA(Y)
        & LEFTOF(X,X2) & LEFTOF(Y,Y2) & NOT CHUNKENDR(X,C1)
        & NOT CHUNKENDR(Y,C2)
    => EQCHUNKTEST(C1,C2,X2,Y2) & NEGATE(1);
V32: "THEY MATCH" = EQCHUNKTEST(C1,C2,X,Y) & EQTHEY(X) & EQTHE(Y)
        & LEFTOF(X,W) & LEFTOF(Y,Z) & LEFTOF(Z,V)
    => EQCHUNKTEST(C1,C2,W,V) & NEGATE(1);
V33: "THE-SKIP" = EQCHUNKTEST(C1,C2,X,Y) & WORDEQ(X,XW) & EQTHE(Y)
        & LEFTOF(Y,Z) & WORDEQ(Z,XW)
    => EQCHUNKTEST(C1,C2,X,Z) & NEGATE(1);
V33R: "THE-SKIP" = EQCHUNKTEST(C1,C2,X,Y) & WORDEQ(Y,YW) & EQTHE(X)
        & LEFTOF(X,W) & WORDEQ(W,YW)
    => EQCHUNKTEST(C1,C2,W,Y) & NEGATE(1);

V34: "SING+PL" = EQCHUNKTEST(C1,C2,X,Y) & WORDEQ(X,XW) & NOT WORDEQ(Y,XW)
        & EQTHE(Y) & STRINGEQ('(NUMBER OF),Y,Z) & ISPLURAL(Z,XW)
        & NOT CHUNKENDR(X,C1) & NOT CHUNKENDR(Z,C2) & LEFTOF(Z,V)
        & LEFTOF(X,W)
    => EQCHUNKTEST(C1,C2,W,V) & NEGATE(1);

V35: "FIRST+ONE" = EQCHUNKTEST(C1,C2,X,Y) & EQONE(X) & EQFIRST(Y) & LEFTOF(X,W)
        & LEFTOF(Y,Z) & EQNUMBER(W) & EQNUMBER(Z) & CHUNKENDR(W,C1)
    => EQVARCHUNK(C1,C2) & NEGATE(1);
V36P: "FIRST+ONE OF" = EQCHUNKTEST(C1,C2,X,Y) & EQONE(X) & EQFIRST(Y)
        & LEFTOF(Y,Z) & EQNUMBER(Z) & STRINGEQ('(OF THE),X,W)
        & EQNUMBERS(W) & CHUNKENDR(W,C1)
    => EQVARCHUNK(C1,C2) & NEGATE(1);
V37: "SECOND+OTHER" = EQCHUNKTEST(C1,C2,X,Y) & EQOTHER(X) & EQSECOND(Y)
        & LEFTOF(X,W) & LEFTOF(Y,Z) & EQNUMBER(W) & EQNUMBER(Z)
        & CHUNKENDR(W,C1)
    => EQVARCHUNK(C1,C2) & NEGATE(1);

V40: "FIN VAR EQ TEST" = EQCHUNKTEST(C1,C2,X,Y) & WORDEQ(X,XW) & WORDEQ(Y,XW)
        & CHUNKENDR(X,C1)
    => EQVARCHUNK(C1,C2) & NEGATE(1);
```

```
V50: "EQVAR EXPR" = EQVARCHUNK(C1 C2) & HASEXPR(C2 X) & NUMVARCHUNKS(N)
          & NOT HASEXPR(C1 X)
     -> NEWPLVAR(C1) & VARCLEANUP(C1) & HASEXPR(C1 X)
          & NUMVARCHUNKS(N-1) & NEGATE(0):

V55: "VAR TEST COUNT" = VARCHCOUNT(C1 C2) & CHTESTED(C1 N) & HASCPRIOR(C2 M2)
          & NOT (EXISTS(C3 M3) & ISVARCHUNK(C3) & HASCPRIOR(C3 M3)
               & SATISFIES(M3 M2 '(GREAT M2))
               & NOT CHTCOUNTED(C1 C1) & VMEQ(C1 C3))
     -> CHTESTED(C1 N+1) & CHTCOUNTED(C2 C1) & NEGATE(1 2):
V60: "VAR TEST END" = CHTESTED(C N) & NUMVARCHUNKS(N)
     -> EXISTS(VAR) & NEWOVAR(C) & NEWPLVAR(C) & VARCLEANUP(C)
          & HASEXPR(C VAR) & NEGATE(1):

V65: "VAR CLEANUP 1" = VARCLEANUP(C) & CHTESTED(C N) -> NEGATE(2):
V80: "VAR CLEANUP 2" = VARCLEANUP(C1) & EQCHUNKTEST(C1 C2 X Y) -> NEGATE(2):
V85: "VAR CLEANUP 3" = VARCLEANUP(C1) & VARCHCOUNT(C1 C2) -> NEGATE(2):
V90: "VAR CLEANUP 4" = VARCLEANUP(C1) & CHTCOUNTED(C2 C1) -> NEGATE(2):

END;
```

. . . . . . . . . . . . . . . . . . .

```
% PAGE 9 - FV SCANNING %

EXPR STUF() BEGIN     FSMACRO(STUDNM);

F5: "WHAT ARE FV" = FVSCAN(X) & EQWHAT(X) & ISSCANFV(C) & CHUNKENDL(X,C)
          & LEFTOF(X,Y) & EQARE(Y) & LEFTOF(Y,Z)
     -> CHUNKENDL(Z,C) & RTQMDMAGOTMS(C) & NEGATE(1 0):
F15: "& QM FV &" = FVSCAN(X) & RTQMDMAGOTMS(C) & EQAND(X)
          & LEFTOF(W,X) & LEFTOF(X,Y)
     -> ISVARCHUNK(C) & CHUNKENDR(W,C) & EXISTS(C) & NEWFV(C)
          & ISFV(C) & RTQMDMAGOTMS(C) & CHUNKENDL(Y,C) & NEGATE(1 2):
F20: "& QM FV 2" = FVSCAN(X) & RTQMDMAGOTMS(C) & EQQMARK(X) & LEFTOF(W,X)
     -> ISVARCHUNK(C) & CHUNKENDR(W,C) & FVSCANEND(X,C) & NEGATE(1 2):

F25: "WHAT IS FV" = FVSCAN(X) & EQWHAT(X) & ISSCANFV(C) & CHUNKENDL(X,C)
          & LEFTOF(X,Y) & EQIS(Y) & LEFTOF(Y,Z)
     -> CHUNKENDL(Z,C) & RTQMGOTMS(C) & NEGATE(1 0):
F35: "QM FV 2" = FVSCAN(X) & RTQMGOTMS(C) & EQQMARK(X) & LEFTOF(W,X)
     -> ISVARCHUNK(C) & CHUNKENDR(W,C) & FVSCANEND(X,C) & NEGATE(1 2):

F40: "HOWM IS FV" = EQHOWM(X) & FVSCAN(X) & ISSCANFV(C) & CHUNKENDL(X,C)
          & LEFTOF(X,Y) & LEFTOF(Y,Z) & EQEQUALS(Z) & LEFTOF(Z,W)
          & EQIT(W) & LEFTOF(W,V)
     -> CHUNKENDL(V,C) & RTQMGOTMS(C) & ISANSUNIT(Y) & NEGATE(2 0):

F45: "HOWM DO HAVE FV" = FVSCAN(X) & EQHOWM(X) & ISSCANFV(C)
          & LEFTOF(X,Y) & CHUNKENDL(X,C) & LEFTOF(Y,Z) & EQDO(Z)
     -> STRINGINS('(THE NUMBER OF) X,Y) & RTDOGOTMS(C)
          & CHUNKENDL(Z,C) & NEGATE(0 5):
F48: "DO END" = RTDOGOTMS(C) & FVSCAN(X) & LEFTOF(X,Y) & EQDO(Y)
          & LEFTOF(Y,Z)
     -> RTQMGOTMS(C) & LEFTOF(X,Z) & NEGATE(ALL,0):

F50: "HOWM DOES HAVE FV" = FVSCAN(X) & EQHOWM(X) & ISSCANFV(C)
          & LEFTOF(X,Y) & CHUNKENDL(X,C) & LEFTOF(Y,Z) & EQDOES(Z)
     -> STRINGINS('(THE NUMBER OF) X,Y) & RTDOESGOTMS(C)
          & CHUNKENDL(Z,C) & NEGATE(0 5):
F52: "DOES END" = RTDOESGOTMS(C) & FVSCAN(X) & LEFTOF(X,Y) & EQDOES(Y)
          & LEFTOF(Y,Z)
     -> RTHAVEGOTMS(C) & LEFTOF(X,Z) & NEGATE(ALL,0):
F55: "HAVE - HAS" = FVSCAN(X) & RTHAVEGOTMS(C) & EQHAVE(X)
     -> EQHAS(X) & WORDEQ(X,'HAS) & NOT WORDEQ(X,'HAVE) & RTQMGOTMS(C)
          & NEGATE(ALL):

F60: "FIND FV" = EQFIND(X) & FVSCAN(X) & ISSCANFV(C) & CHUNKENDL(X,C)
          & LEFTOF(X,Y)
     -> CHUNKENDL(Y,C) & RTANDPERGOTMS(C) & NEGATE(2 0):
F70: "& . . ." = FVSCAN(X) & RTANDPERGOTMS(C) & EQPERIOD(X) & LEFTOF(W,X)
     -> ISVARCHUNK(C) & CHUNKENDR(W,C) & FVSCANEND(X,C) & NEGATE(1 2):
F75: "& . &" = FVSCAN(X) & RTANDPERGOTMS(C) & EQAND(X)
          & LEFTOF(W,X) & LEFTOF(X,Y)
     -> ISVARCHUNK(OC) & CHUNKENDR(W,OC) & EXISTS(C) & NEWFV(C)
          & ISFV(C) & RTANDPERGOTMS(C) & CHUNKENDL(Y,C) & NEGATE(1 2):

F80: "NEW FV" = ISFV(FV) & MXCPRIOR(M) & NOT (EXISTS(N) & HASCPRIOR(FV,N))
     -> HASCPRIOR(FV,M+1) & MXCPRIOR(M+1) & NEGATE(2):
```

```
END;
```

. . . . . . . . . . . . . . . . . . . . . . .

```
% PAGE 10 - AGE PROBLEM HEURISTICS %

EXPR STUAH(): BEGIN     FSMACRO(STUDNM);

A1: "AGE SCAN1" = ASCAN(P) & EQAS(X) & LEFTOF(X,Y) & EQOLD(Y)
          & LEFTOF(Y,Z) & EQAS(Z)
     -> AGEPROB(P) & AGEREFCNT(1) & NEGATE(1):
A2: "AGE SCAN2" = ASCAN(P) & EQAGE(X) -> AGEPROB(P) & AGEREFCNT(1) & NEGATE(1):
A3: "AGE SCAN3" = ASCAN(P) & EQYEARS(X) & LEFTOF(X,Y) & EQOLD(Y)
     -> AGEPROB(P) & AGEREFCNT(1) & NEGATE(1):
A11: "DEL AS OLD AS" = TFASCAN(V?.1) & STRINGEQ('(AS OLD AS) X,Z)
     -> MODLEN(-3) & TFOUT(V?.1,Z) & LEFTOF(X,Z) & NEGATE(ALL):
A12: "DEL YEARS OLD" = TFASCAN(V?.1) & STRINGEQ('(YEARS OLD) X,Z)
     -> MODLEN(-2) & TFOUT(V?.1,Z) & LEFTOF(X,Z) & NEGATE(ALL):
A15: "WILL BE WHEN" = TFASCAN(V?.1) & STRINGEQ('(WILL BE WHEN) X,Z)
     -> MODLEN(4) & TFOUTLEN(V?.1,X,-1)
          & DELAYEXPND(STRINGINS('(IN (GENSYM) YEARS ?, IN (GENSYM) YEARS) X,Z))
          & NEGATE(ALL):
A17: "WAS WHEN" = TFASCAN(V?.1) & STRINGEQ('(WAS WHEN) X,Z)
     -> MODLEN(5) & TFOUTLEN(V?.1,X,-1)
          & DELAYEXPND(STRINGINS('((GENSYM) YEARS AGO ?, (GENSYM)
               YEARS AGO) X,Z))
          & NEGATE(ALL):
A18: "WAS - IS" = TFASCAN(X) & EQWAS(X) & LEFTOF(X,Y) & NOT EQWHEN(Y)
     -> TFSCAN(X) & TFSCANFIN(N,4) & EQIS(X) & WORDEQ(X,'IS) & NOT WORDEQ(X,'WAS)
          & NEGATE(1 2):
A19: "WILL BE - IS" = TFASCAN(V?.1) & STRINGEQ('(WILL BE) X,Z) & NOT EQWHEN(Z)
     -> MODLEN(-1) & TFSCAN(V?.1) & TFSCANFIN(V?.1) & EQIS(V?.1)
          & WORDEQ(V?.1,'IS) & NOT WORDEQ(V?.1,'WILL) & LEFTOF(V?.1,Z)
          & NEGATE(ALL,-2):
A20: "IS NOW" = TFASCAN(X) & STRINGEQ('(IS NOW) X,Z)
     -> MODLEN(1) & STRINGINS('(S AGE NOW) X,Z)
          & NEGATE(ALL):

A24: "AGE OP+" = TFASCAN(X) & CHUNKENDL(X,C) & EQIN(X)
          & LEFTOF(X,Y) & LEFTOF(Y,Z) & EQYEARS(Z) & LEFTOF(Z,W)
     -> MODLEN(-3) & TFOUT(X,W) & AGEOP(X,C) & NEGATE(1):
A26: "AGE OP-" = TFASCAN(X) & CHUNKENDL(X,C) & LEFTOF(X,Y)
          & EQYEARS(Y) & LEFTOF(Y,Z) & EQAGO(Z) & LEFTOF(Z,W)
     -> MODLEN(-3) & TFOUT(X,W) & AGEOP(X,C) & NEGATE(1):
A28: "AGE OP+2" = TFASCAN(X) & CHUNKENDL(X,C)
          & STRINGEQ('(YEARS FROM NOW) X,Z)
     -> MODLEN(-4) & TFOUT(X,Z) & AGEOP(X,C) & NEGATE(1):

A31: "AGE +S" = TFASCAN(X) & EQAGE(X) & LEFTOF(X,Y) & EQIN(Y) & LEFTOF(Y,N)
          & LEFTOF(N,Z) & EQYEARS(Z) & LEFTOF(Z,W)
     -> MODLEN(-1) & STRINGINS('(PLUSS) X,N) & LEFTOF(N,W) & NEGATE(ALL,-2):
A32: "AGE +S2" = TFASCAN(X) & EQAGE(X) & LEFTOF(X,Y) & STRINGEQ('(YEARS FROM NOW),Y,Z)
     -> MODLEN(-2) & STRINGINS('(PLUSS) X,Y) & LEFTOF(Y,Z) & NEGATE(ALL,-2):
A34: "AGE -S" = TFASCAN(X) & EQAGE(X) & LEFTOF(X,Y) & STRINGEQ('(YEARS AGO),Y,Z)
     -> MODLEN(-1) & STRINGINS('(MINUSS) X,Y) & LEFTOF(Y,Z) & NEGATE(ALL,-2):
A35: "AGE NOW" = TFASCAN(X) & EQAGE(X) & LEFTOF(X,Y) & EQNOW(Y) & LEFTOF(Y,Z)
     -> MODLEN(-1) & LEFTOF(X,Z) & NEGATE(ALL,-2):

A38: "AGE OP NEG" = TFASCAN(X) & EQAGE(X) & LEFTOF(X,Y) & LEFTOF(Y,Z)
          & LEFTOF(Z,V) & LEFTOF(V,W)
          & NOT( EQWILL(Y) & EQBE(Z) & EQWHEN(V) )
          & NOT( EQWAS(Y) & EQWHEN(Z) )
          & NOT( EQYEARS(Z) & EQFROM(V) & EQNOW(W) )
          & NOT EQNOW(Y) & NOT( EQIN(Y) & EQYEARS(Y) )
          & NOT( EQYEARS(Z) & EQAGO(V) )
          & ISSCANCHUNK(F) & AGEOP(O,C) & WORDEQ(O,OW)
     -> AGEOPNEED(X,Y,D,'OW') & NEGATE(1 3):

A41: "AGE OP COLL-" = AGEOPNEED(X,Y,O,L) & SATISFIES(L,LENGTH(L) ?,GREAT 2)
          & LEFTOF(O,P) & NOT( EQFROM(O) & EQNOW(P) )
     -> MODLEN(LENGTH L) & TFASCAN(X) & DELAYEXPND(STRINGINS(L,X,Y))
          & NEGATE(1):
A42: "AGE OP COLL+" = AGEOPNEED(X,Y,O,L) & SATISFIES(L,LENGTH(L) ?,LESS 3)
          & LEFTOF(O,P) & WORDEQ(P,PW)
     -> AGEOPNEED(X,Y,P,L @ 'PW') & NEGATE(1):
A43: "AGE OP COLL+" = AGEOPNEED(X,Y,O,L) & SATISFIES(L,LENGTH(L) EQ 3)
          & LEFTOF(O,P) & EQFROM(O) & EQNOW(P) & WORDEQ(P,PW)
     -> AGEOPNEED(X,Y,P,L @ 'PW') & NEGATE(1):
```

```
A50: "THEIR AGES" = TEASCAN(V7.1) & STRINGEQ(THEIR AGES)X2)
     -> THEIRREF(X.Z) & TEQUITDELAY(V7.1.X) & NEGATE(ALL);
A51: "START THEIR COLL" = THEIRREF(X.Y) & AGEREF(A.P) & NOT THEIRCOLLD(A.X)
          & NOT( EXISTS(A2.P2) & AGEREF(A2.P2) & VNLQ(A.A2)
               & NOT THEIRCOLLD(A2.X) & SATISFIES2(P.P2.P2 ?•LESS P) )
          & WORDEQ(A.AW)
     -> THEIRCOLL(X.Y.A.A. AW ) & THEIRREF(X.Y);
A52: "THEIR COLL +" = THEIRCOLL(X.Y.T.A.L) & NOT EQAGE(A) & LEFTOF(A.B)
          & WORDEQ(B.BW)
     -> THEIRCOLL(X.Y.T.B. @ BW.B) & NEGATE(1);
A53: "THEIR COLL -" = THEIRCOLL(X.Y.T.A.L) & EQAGE(A) -> THEIRCOLLD(T.X);
A56: "THEIR COLLD" = THEIRCOLLD(A.X) & THEIRCOLL(X.Y.A.B.L) & THEIRREFL(L2)
     -> THEIRREFL(L2 @ '(AND) @ L) & NEGATE(2.3);
A57: "THEIR COLLD 1" = THEIRCOLLD(A.X) & NOT( EXISTS(L) & THEIRREFL(L) )
          & THEIRCOLL(X.Y.A.B.L) )
     -> THEIRREFL(L) & NEGATE(3);
A59: "THEIR COLL E" = THEIRREFL(L) & THEIRREF(X.Y)
          & NOT( EXISTS(A.P) & AGEREF(A.P) & NOT THEIRCOLLD(A.X) )
     -> MODLEN(LENGTH(L) - 2) & DELAYEXPND(STRINGINS(L.X.Y)) & NEGATE(1.2);

A61: "PERSON - AGE" = ISPERSON(X) & AGEPROB(P) & LEFTOF(X.Y) & NOT EQS(Y)
     -> MODLEN(2) & ISPERSON(X) & STRINGINS(IS AGE.X.Y) & NEGATE(3);
A62: "PERSON. AGE" = ISPERSON(X) & AGEPROB(P) & LEFTOF(X.Y) & EQS(Y)
          & LEFTOF(W.X) & NOT EQS(W)
     -> AGEREFCHK(X);
A63: "AGE REF CHK" = AGEREFCHK(X) & AGEREF(Y.N)
     -> AGECOMP(X.Y) & AGECOMPLIM(X) & NEGATE(1);
A631: "AGE REF 1" = AGEREFCHK(X) & NOT( EXISTS(Y.N) & AGEREF(Y.N) )
     -> AGECOMPLIM(X) & AGECOMP(X.X) & NEGATE(1);
A64: "AGE REF NEW" = AGECOMPLIM(X) & AGEREFCNT(N)
     -> AGECOMPREM(X) & AGEREF(X.N) & AGEREFCNT(N+1) & NEGATE(ALL);
A66: "AGECOMPREM" = AGECOMPREM(X) & AGECOMP(Y.Z) -> NEGATE(ALL);
A67: "AGEREF 1ST +" = AGECOMP(X.Y) & WORDEQ(X.XW) & WORDEQ(Y.XW) & NOT EQAGE(X)
          & LEFTOF(X.W) & LEFTOF(Y.Z)
     -> AGECOMP(W.Z) & NEGATE(1);
A68: "AGEREF 1ST LINE" = AGECOMP(X.Y) & WORDEQ(X.XW) & WORDEQ(Y.YW) & VNLQ(XW.YW)
          & EQS(Y) & LEFTOF(Y.Z) & EQAGE(Z) & AGECOMPLIM(V)
     -> AGECOMPREM(V) & NEGATE(B);
A69: "AGEREF 1ST LINE" = AGECOMP(X.Y) & EQAGE(X) & EQAGE(Y) & AGECOMPLIM(Z)
     -> AGECOMPREM(Z) & NEGATE(A);

A71: "AGE PRON" = ISPRON(X) & AGEREF(Y.N) & SATISFIES(N.N EQ 1) & WORDEQ(Y.YW)
     -> AGEPRONCOL(Y.YW.X);
A73: "AGE PRON COL +" = AGEPRONCOL(P.L.O) & LEFTOF(P.Q) & NOT EQAGE(Q)
          & WORDEQ(Q.QW)
     -> AGEPRONCOL(Q.L @ QW.O) & NEGATE(1);
A75: "AGE PRON COLE" = AGEPRONCOL(P.L.O) & LEFTOF(P.Q) & EQAGE(Q)
          & LEFTOF(A.O) & LEFTOF(O.B)
     -> MODLEN(LENGTH(L) ) & TEQUITDELAY(O.A)
          & DELAYEXPND(STRINGINS(L @ '(AGE) A.B)) & NEGATE(1.4.5)
          & NOT TGSCANTIN(O);

A77: "TEQUIT DELAY" = TEQUITDELAY(O.A) & LEFTOF(A.B) -> TEQUIT(O.B) & NEGATE(1);

A81: "AGE PRON" = ISPOSSPRON(X) & AGEREF(Y.N) & SATISFIES(N.N EQ 1) & WORDEQ(Y.YW)
     -> AGEPOSSCOL(Y. YW .X);
A83: "AGE POSS COL +" = AGEPOSSCOL(P.L.O) & LEFTOF(P.Q) & NOT EQAGE(Q)
          & WORDEQ(Q.QW)
     -> AGEPOSSCOL(Q.L @ QW.O) & NEGATE(1);
A85: "AGE POSS COLE" = AGEPOSSCOL(P.L.O) & LEFTOF(P.Q) & EQAGE(Q)
          & LEFTOF(A.O) & LEFTOF(O.B)
     -> MODLEN(LENGTH(L) - 1) & TEQUITDELAY(O.A)
          & DELAYEXPND(STRINGINS(L.A.B)) & NEGATE(1.4.5) & NOT TGSCANTIN(O);
```

. . . . . . . . . . . . . . . . . . . . . .

```
% PAGE 11 - ANSWER-BUILDING PRODS %

B1: "EV LIST" = ANSWERBUILD(P)
     -> EVLIST(P.'NIL) & ANSUNITCHK(P) & ANSWERBUILD2(P) & NEGATE(1);
B2: "EV LIST ADD" = EVLIST(P.L) & ISEV(E) & HASCPRIOR(E.N) & HASEXPR(E.Z)
          & NOT( EXISTS(E2.N2) & HASCPRIOR(E2.N2) & ISEV(E2)
               & SATISFIES2(N.N2.N2 ?•GREAT N) )
     -> EVLIST(P.'E CONS CAR L ') & NEGATE(1.2);
B3: "UNIT CHK" = ANSUNITCHK(X) & ISANSUNIT(Y) & ISPLURAL(Y.YW)
     -> ISANSUNIT(YW) & NEGATE(1.2);
B5: "VARS REPR" = ANSWERBUILD2(P) & ISVARCHUNK(V) & HASCPRIOR(V.N) & HASEXPR(V.E)
          & NOT( EXISTS(V2.E2.N2) & HASCPRIOR(V2.N2) & ISVARCHUNK(V2)
               & SATISFIES2(N2.N.N2 ?•LESS N)
               & HASEXPR(V2.E2) & NOT( EXISTS(R) & HASREPR(E2.R) ) )
          & NOT( EXISTS(R) & HASREPR(E.R) )
     -> BUILDREPR(V) & ANSWERBUILD2(P);
B6: "VAR REPR ST" = BUILDREPR(V) & HASEXPR(V.E) & CHUNKEND( (X.V) & WORDEQ(X.W)
     -> WCOLLECT(V.E.X) & HASREPR(E.'(W)) & NEGATE(1);
B8: "VAR REPR SCAN" = WCOLLECT(V.E.X) & HASREPR(E.L) & NOT CHUNKENDR(X.V)
          & LEFTOF(X.Y) & WORDEQ(Y.W)
     -> WCOLLECT(V.E.Y) & HASREPR(E.L @ '(W)) & NEGATE(1.2);

END;
```

. . . . . . . . . . . . . . . . . . . . . .

```
% PAGE 12 - INFO PRODS %

EXPR STUI();BEGIN

I1: "INIT INFO" = PROBLEM(P) & STRLENGTH(L) & NOT( EXISTS(N) & SPACESIZEN(N) )
     -> NEWSIZE(P) & SPACESIZEN(O) & NUMVARCHUNKS(O)
          & SPACESIZES(O.'EQNS.'EVS.'OPS.'VARS)
          & PROBEQNS( (L-3)/4 .0) % #INDEF #DEF %
          & PROBEVS( (L-1)/4 .0) % #INDEF #FOUND %
          & PROBOPS( (L-7)/2 .0.0) % #INDEF #DEF #PLACED %
          & PROBVARS( (L-3)/2 .0.0.0):
               % #INDEF #CN(OPS.EQNS) #PLACED #DISTINCT %

I3: "NEW SIZE" = NEWSIZE(P) & SPACESIZEN(N) & PROBVARS(V1.V2.V3.V4)
          & PROBEVS(E1.E2) & PROBOPS(O1.O2.O3) & PROBEQNS(E1.E2)
     -> SPACESIZES(N+1. 'E1.E2'. 'E1.E2'. 'O1.O2.O3'. 'V1.V2.V3.V4' )
          & SPACESIZEN(N+1) & NEGATE(1.2);
I5: "NEW LEN" = MODLEN(N) & SATISFIES(N.N NEQ O) & STRLENGTH(L) & PROBEVS(E1.E2)
          & PROBVARS(V1.V2.V3.V4) & SATISFIES(E2.E2 EQ O)
          & PROBOPS(O1.O2.O3) & PROBEQNS(E1.E2)
     -> NEWSIZE(N) & STRLENGTH(L+N) & NEGATE(ALL)
          & PROBEVS( MAX(V4.(L+N-1)/4) .E2) & PROBVARS( (L+N-3)/2 .V2.V3.V4)
          & PROBOPS( (L+N-7)/2 .O2.O3) & PROBEQNS( (L+N-3)/4 .E2);
I7: "NEW LEN C" = MODLENC(X) & CHUNKLEN(N) & SATISFIES(N.N NEQ O)
     -> MODLEN(-N) & CHUNKLEN(O) & NEGATE(ALL);

I11: "NEW OP" = NEWOP(X) & WORDEQ(X.XW) & SPACESIZEN(N)
          & PROBOPS(O1.O2.O3) & PROBVARS(V1.V2.V3.V4)
     -> PROBOPS(O1.O2+1.O3) & PROBVARS(V1.V2+1.V3.V4) & MODLENC(X)
          & DEFOPLIST(N+1.XW) & NEGATE(1.4.5);
I13: "NEW REF OP" = NEWREFOP(X) & SPACESIZEN(N)
          & PROBOPS(O1.O2.O3) & PROBVARS(V1.V2.V3.V4)
     -> NEWSIZE(X) & PROBOPS(O1.O2+1.O3) & PROBVARS(V1.V2+1.V3.V4)
          & DEFOPLIST(N+1.X) & NEGATE(1.3.4);
I15: "NEW PL OP" = NEWPLOP(C) & HASOP(C.O) & SPACESIZEN(N) & PROBOPS(O1.O2.O3)
          & NOT SATISFIES(O.O EQ 'EQUAL)
     -> NEWSIZE(C) & PROBOPS(O1.O2.O3+1) & PLACOPLIST(N+1.O) & NEGATE(1.4);
I17: "NEW PL =" = NEWPLOP(C) & HASOP(C.O) & SATISFIES(O.O EQ 'EQUAL)
          & SPACESIZEN(N)
     -> PLACOPLIST(N+0.5.O) & NEGATE(1);

I21: "NEW EQN" = NEWEQN(X) & PROBEQNS(E1.E2) & PROBVARS(V1.V2.V3.V4)
     -> NEWSIZE(X) & PROBEQNS(E1.E2+1) & PROBVARS(V1.V2+2.V3.V4) & NEGATE(ALL);
I31: "NEW VAR DIST" = NEWDVAR(X) & PROBVARS(V1.V2.V3.V4) & PROBEVS(E1.E2)
     -> NEWSIZE(X) & PROBVARS(V1.V2.V3.V4+1) & NEGATE(ALL)
          & PROBEVS(MAX(E1.V4+1).E2);
I33: "NEW VAR PL" = NEWPLVAR(X) & PROBVARS(V1.V2.V3.V4)
          & PROBEVS(E1.E2) & SATISFIES(E2.E2 EQ O)
     -> NEWSIZE(X) & PROBVARS(V1.V2.V3+1.V4) & NEGATE(1.2);
I41: "NEW EV 1" = NEWEV(X) & PROBEVS(E1.E2) & SATISFIES(E2.E2 EQ O)
          & PROBVARS(V1.V2.V3.V4) & PROBEQNS(E1.E2) & PROBOPS(O1.O2.O3)
     -> NEWSIZE(X) & NEGATE(ALL) & PROBEVS(V4+1.1) & PROBVARS(O.V2.V3.V4)
          & PROBEQNS(O.E2) & PROBOPS(O.O2.O3);
I43: "NEW EV" = NEWEV(X) & PROBEVS(E1.E2) & SATISFIES(E2.E2 NEQ O)
     -> NEWSIZE(X) & PROBEVS(E1+1.E2+1) & NEGATE(1.2);

END;END.
```

. . . . . . . . . . . . . . . . . . . . . .

```
BEGIN  % STUDNT EXAMPLE MODULES %          % FROM FILE STUXS %
```

```
EXPR STUDX1();BEGIN    PSMACRO(STUDNM);

X1: TEST1(X) ← INITPROB(X,'(A PLUS B IS 5 ?, B IS 3 ?, FIND A ?));

X2: TEST2(X) ← INITPROB(X,'(A FIRST NUMBER PLUS 6 IS EQUAL TO
      A SECOND NUMBER ?, TWICE THE FIRST NUMBER IS
      THREE TIMES ONE HALF OF THE SECOND NUMBER ?.
      WHAT ARE THE FIRST NUMBER AND THE SECOND NUMBER ??));

X3: TEST3(P) ← INITPROB(P,'(A NUMBER IS MULTIPLIED BY 6 ?.
      THIS PRODUCT IS INCREASED BY 44 ?.
      THIS RESULT IS 68 ?, FIND THE NUMBER ?));

END;

EXPR STUDX2();BEGIN    PSMACRO(STUDNM);

X4: TEST4(P) ← INITPROB(P,'(IF THE NUMBER OF CUSTOMERS TOM GETS IS
      TWICE THE SQUARE OF 20 PER CENT OF
      THE NUMBER OF ADVERTISEMENTS HE RUNS ?,
      AND THE NUMBER OF ADVERTISEMENTS HE RUNS IS 45 ?,
      WHAT IS THE NUMBER OF CUSTOMERS TOM GETS ??));

X5: TEST5(P) ← INITPROB(P,'(THE SUM OF LOIS SHARE OF SOME MONEY
      AND BOB S SHARE IS $ 4.50 ?,
      LOIS SHARE IS TWICE BOB S ?, FIND BOB S AND LOIS SHARE ?.));

X6: TEST6(P) ← INITPROB(P,'(MARY IS TWICE AS OLD AS ANN WAS
      WHEN MARY WAS AS OLD AS ANN IS NOW ?.
      IF MARY IS 24 YEARS OLD ?, HOW OLD IS ANN ??));

END;

EXPR STUDX3();BEGIN    PSMACRO(STUDNM);

X7: TEST7(P) ← INITPROB(P,'(THE SUM OF THE PERIMETER OF A RECTANGLE AND
      THE PERIMETER OF A TRIANGLE IS 24 INCHES ?.
      IF THE PERIMETER OF THE RECTANGLE IS
      TWICE THE PERIMETER OF THE TRIANGLE ?,
      WHAT IS THE PERIMETER OF THE TRIANGLE ??));

X8: TEST8(P) ← INITPROB(P,'(THE PRICE OF A RADIO IS 69.70 DOLLARS ?.
      IF THIS PRICE IS 15 PER CENT LESS THAN THE MARKED PRICE ?,
      FIND THE MARKED PRICE ?.));

X9: TEST9(P) ← INITPROB(P,'(BILL IS ONE HALF OF HIS FATHER S AGE
      8 YEARS AGO ?.
      IN 20 YEARS HE WILL BE 2 YEARS OLDER THAN HIS FATHER IS NOW ?.
      HOW OLD ARE BILL AND HIS FATHER ??));

END;

EXPR STUDX4();BEGIN    PSMACRO(STUDNM);

X10: TEST10(P) ← INITPROB(P,'(BILL S FATHER S UNCLE IS TWICE AS OLD
      AS BILL S FATHER ?. 2 YEARS FROM NOW BILL S FATHER WILL BE
      3 TIMES AS OLD AS BILL ?.
      THE SUM OF THEIR AGES IS 92 ?, FIND BILL S AGE ?.));

X11: TEST11(P) ← INITPROB(P,'(TOM HAS TWICE AS MANY FISH AS MARY
      HAS GUPPIES ?. IF MARY HAS 3 GUPPIES ?,
      HOW MANY FISH DOES TOM HAVE ??));

X12: TEST12(P) ← INITPROB(P,'(IF 1 SPAN EQUALS 9 INCHES ?,
      AND 1 FATHOM EQUALS 6 FEET ?,
      HOW MANY SPANS EQUALS 1 FATHOM ??));
END;

EXPR STUDX5();BEGIN    PSMACRO(STUDNM);

X13: TEST13(P) ← INITPROB(P,'( THE NUMBER OF SOLDIERS THE RUSSIANS HAVE
      IS ONE HALF OF THE NUMBER OF GUNS THEY HAVE ?.
      THE NUMBER OF GUNS THEY HAVE IS 7000 ?.
      HOW MANY SOLDIERS DO THEY HAVE ??));

X14: TEST14(P) ← INITPROB(P,'( THE NUMBER OF STUDENTS WHO PASSED THE
      ADMISSIONS TEST IS 10 PER CENT OF THE TOTAL NUMBER OF STUDENTS IN
      THE HIGH SCHOOL ?. IF THE NUMBER OF SUCCESSFUL CANDIDATES IS 72
      ?, WHAT IS THE NUMBER OF STUDENTS IN THE HIGH SCHOOL ??));

X15: TEST15(P) ← INITPROB(P,'( THE DISTANCE FROM NEW YORK TO LOS
      ANGELES IS 3000 MILES ?. IF THE AVERAGE SPEED OF A JET PLANE
      IS 600 MILES PER HOUR ?, FIND THE TIME IT TAKES
      TO TRAVEL FROM NEW YORK TO LOS ANGELES BY JET ?.));

END;

EXPR STUDX6();BEGIN    PSMACRO(STUDNM);

X16: TEST16(P) ← INITPROB(P,'( THE COST OF A BOX OF MIXED NUTS IS
      THE SUM OF THE COST OF THE ALMONDS IN THE BOX AND THE COST OF
      THE PECANS IN THE BOX ?. FOR A LARGE BOX THIS COST IS $
      3.500 ?. THE WEIGHT ?, IN POUNDS ?, OF A BOX OF MIXED NUTS IS
      THE SUM OF THE NUMBER OF POUNDS OF ALMONDS IN THE BOX AND
      THE NUMBER OF POUNDS OF PECANS IN THE BOX ?.
      THIS LARGE BOX WEIGHS 3 POUNDS ?.
      THE COST OF ALMONDS PER POUND OF ALMONDS IS $ 1 ?,
      AND THE COST OF PECANS PER POUND OF PECANS IS $ 1.500 ?.
      FIND THE COST OF THE ALMONDS IN THE BOX AND THE COST OF THE
      PECANS IN THE BOX ?.));

X17: TEST17(P) ← INITPROB(P,'( THE SUM OF TWO NUMBERS IS 96 ?,
      AND ONE NUMBER IS 16 LARGER THAN THE OTHER NUMBER ?.
      FIND THE TWO NUMBERS ?.));

X18: TEST18(P) ← INITPROB(P,'( 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 ??));

END;

EXPR STUDX7();BEGIN    PSMACRO(STUDNM);

X19: TEST19(P) ← INITPROB(P,'( THE DAILY COST OF LIVING FOR A GROUP
      IS THE OVERHEAD COST PLUS THE RUNNING COST FOR EACH PERSON
      TIMES THE NUMBER OF PEOPLE IN THE GROUP ?.
      THIS COST FOR ONE GROUP EQUALS $ 100 ?, AND THE NUMBER OF
      PEOPLE IN THE GROUP IS 40 ?.
      IF THE OVERHEAD COST IS 10 TIMES THE RUNNING COST ?,
      FIND THE OVERHEAD AND THE RUNNING COST FOR EACH PERSON ?.));

X20: TEST20(P) ← INITPROB(P,'( THE RUSSIAN ARMY HAS 6 TIMES AS MANY
      RESERVES IN A UNIT AS IT HAS UNIFORMED SOLDIERS ?.
      THE PAY FOR RESERVES EACH MONTH IS 50 DOLLARS TIMES THE NUMBER
      OF RESERVES IN THE UNIT ?. AND THE AMOUNT SPENT ON THE REGULAR
      ARMY EACH MONTH IS $ 150 TIMES THE NUMBER OF UNIFORMED
      SOLDIERS ?.
      THE SUM OF THIS LATTER AMOUNT AND THE PAY FOR RESERVES EACH
      MONTH EQUALS $ 45000 ?. FIND THE NUMBER OF RESERVES IN A UNIT
      THE RUSSIAN ARMY HAS AND THE NUMBER OF UNIFORMED
      SOLDIERS IT HAS ?.));

X21: TEST21(P) ← INITPROB(P,'( THE SUM OF TWO NUMBERS IS TWICE THE
      DIFFERENCE BETWEEN THE TWO NUMBERS ?.
      THE FIRST NUMBER EXCEEDS THE SECOND NUMBER BY 5 ?.
      FIND THE TWO NUMBERS ?.));

END;

EXPR STUDX8();BEGIN    PSMACRO(STUDNM);

X22: TEST22(P) ← INITPROB(P,'( THE SUM OF TWO NUMBERS IS 111 ?.
      ONE OF THE NUMBERS IS CONSECUTIVE TO THE OTHER NUMBER ?.
      FIND THE TWO NUMBERS ?.));

X23: TEST23(P) ← INITPROB(P,'( THE SUM OF THREE NUMBERS IS 9 ?.
      THE SECOND NUMBER IS 3 MORE THAN 2 TIMES THE FIRST NUMBER ?.
      THE THIRD NUMBER EQUALS THE SUM OF THE FIRST TWO NUMBERS ?.
      FIND THE THREE NUMBERS ?.));

X24: TEST24(P) ← INITPROB(P,'( THE SUM OF THREE NUMBERS IS 100 ?.
      THE THIRD NUMBER EQUALS THE SUM OF THE FIRST TWO NUMBERS ?.
      THE DIFFERENCE BETWEEN THE FIRST TWO NUMBERS IS 10 PER CENT OF
      THE THIRD NUMBER ?. FIND THE THREE NUMBERS ?.)) ;

END;

EXPR STUDX9();BEGIN    PSMACRO(STUDNM);

X25: TEST25(P) ← INITPROB(P,'( IF C EQUALS B TIMES B PLUS 1 ?,
      AND B PLUS B EQUALS 3 ?, AND B MINUS B EQUALS 1 ?, FIND C ?.
      ));

X26: TEST26(P) ← INITPROB(P,'( THE SQUARE OF THE DIFFERENCE BETWEEN
      THE NUMBER OF APPLES AND THE NUMBER OF ORANGES ON THE TABLE
```

```
     IS EQUAL TO 9 ?. IF THE NUMBER OF APPLES IS 7 ?.
     FIND THE NUMBER OF ORANGES ON THE TABLE ?.) );

X27: TEST27(P) <- INITPROB(P, '( THE GROSS WEIGHT OF A SHIP IS 20000
     TONS ?. IF ITS NET WEIGHT IS 15000 TONS ?.
     WHAT IS THE WEIGHT OF THE SHIPS CARGO ?? ) );

END; END.
```

## Appendix C. CROSS-REFERENCE OF STUDENT PREDICATES

```
XREF OF STUDENT PREDS

AGECOMP
 LHSUSES A66 A67 A68 A69
 RHSUSES A63 A631 -A66 A67 -A67
AGECOMPLIN
 LHSUSES A64 A68 A69
 RHSUSES A63 A631 -A64 -A68 -A69
AGECOMPREM
 LHSUSES A66
 RHSUSES A64 -A66 A68 A69
AGEOP
 LHSUSES A38
 RHSUSES A24 A26 A28
AGEOPNEED
 LHSUSES A41 A42 A43
 RHSUSES A38 -A41 A42 -A42 A43 -A43
AGEPOSSCOL
 LHSUSES A83 A85
 RHSUSES A81 A83 -A83 -A85
AGEPROB
 LHSUSES S16 D55 D57 A61 A62
 NESTEDL S17
 RHSUSES A1 A2 A3
AGEPRONCOL
 LHSUSES A73 A75
 RHSUSES A71 A73 -A73 -A75
AGEREF
 LHSUSES A51 A63 A71 A81
 NESTEDL A51 A59 A631
 RHSUSES A64
AGEREFCHK
 LHSUSES A63 A631
 RHSUSES A62 -A63 -A631
AGEREFCNT
 LHSUSES A64
 RHSUSES A1 A2 A3 A64 -A64
ANSUNITCHK
 LHSUSES B3
 RHSUSES B1 -B3
ANSWERBUILD
 LHSUSES B1
 RHSUSES S70 -B1
ANSWERBUILD2
 LHSUSES B5
 RHSUSES D1 B5
ASCAN
 LHSUSES A1 A2 A3
 RHSUSES -A1 -A2 -A3
BUILDSEPR
 LHSUSES B6
 RHSUSES B5 -B6
CHKCOUNTED
 LHSUSES V90
 NESTEDL -V55
 RHSUSES V23 V55 -V90
CHKTESTED
 LHSUSES V55 V60 V65
 RHSUSES V5 V55 -V55 -V60 -V65
CHUNKENDL
 LHSUSES S60 T50 P10 P15 M20 M30 M50 M55 -C15 C17 C50 -R2 R4 V10 V75 F5 F25
 F40 F45 F50 F60 A24 A26 A28 B6
 NESTEDL T51 V5
 RHSUSES S10 S40 S65 T50 -T50 M20 -M20 M30 -M30 M50 -M50 M55 -M55 C17 -C17 C22
  C50 -C50 C60 R4 -R4 F5 -F5 F15 F25 -F25 F40 -F40 F45 -F45 F50 -F50 F60 -F60
 F75
CHUNKENDR
 LHSUSES -P20 -P23 -P26 P27 P28 P29 M20 M30 M50 M55 C15 C17 -C50 C52 -R6 -R7
 R8 R9 -R9 -V30 -V31 -V34 V35 V36P V37 V40 -B8
 RHSUSES S40 M20 -M20 M30 -M30 M50 M55 -M55 C15 -C15 C17 -C17 C50 C52 -C52 C60
  R8 -R8 F15 F20 F35 F70 F75
CHUNKLEN
 LHSUSES S13 T50 T52 I7
 RHSUSES S10 S13 -S13 S40 -S40 T50 -T50 T52 -T52 I7 -I7
CSPLIT
 LHSUSES C60
 RHSUSES M10 C2 C5 C8 C10 C17 C22 C25 C52 C55 -C60
DEFOPLIST
 RHSUSES I11 I13
ENDMARK
```

```
   LHSUSES S10 -S65 S70
 EQCHUNKTEST
   LHSUSES V30 V31 V32 V33 V33R V34 V35 V36P V37 V40 V80
   RHSUSES V25 V30 -V30 V31 -V31 V32 -V32 V33 -V33 V33R -V33R V34 -V34 -V35
   -V36P -V37 -V40 -V80
 EQIS
   LHSUSES T2 -T19 -T20 C30 P2 M10 F25 A20
   RHSUSES T28 T32 M20 M30 M50 M55 A18 A19 -A20
 EQVARCHUNK
   LHSUSES V23 V50
   NESTEDL V24 V25
   RHSUSES V35 V36P V37 V40
 EQVARPEND
   LHSUSES V25 V26
   RHSUSES V23 V24 -V25 -V26
 EVLIST
   LHSUSES B2
   RHSUSES B1 B2 -B2
 EVSCAN
   LHSUSES F5 F15 F20 F25 F35 F40 F45 F48 F50 F52 F55 F60 F70 F75
   RHSUSES -S15 S25 -F5 -F15 -F20 -F25 -F35 -F40 -F48 -F52 -F55 -F60 -F70 -F75
 EVSCANEND
   LHSUSES S65 S70
   RHSUSES -S65 -S70 F20 F35 F70
 HASCPRIOR
   LHSUSES P10 P15 M20 M30 M50 M55 C70 C75 C78 V55 B2 B5
   NESTEDL P10 P15 V55 F80 B2 B5
   RHSUSES S40 S65 M20 -M20 M30 -M30 M50 -M50 M55 -M55 C60 F80
 HASEXPR
   LHSUSES C70 C75 C78 C90 V10 V15 V50 -V50 B2 B5 B6
   NESTEDL V5 B5
   RHSUSES C70 C75 C78 V10 V15 V50 V60
 HASIS
   LHSUSES M10 C2 C5 C8
   RHSUSES P35 -M10
 HASOP
   LHSUSES C70 C75 C78 C90 I15 I17
   RHSUSES M10 C2 C5 C8 C10 C17 C22 C25 C52 C55
 HASOPO
   LHSUSES C55
   RHSUSES P70 -C55
 HASOP1
   LHSUSES C25
   RHSUSES P50 -C25
 HASOP2
   LHSUSES C10
   RHSUSES P40 -C10
 HASPREC
   LHSUSES S20 S30 P20 P23 P27 P28
   NESTEDL S25 P26 P29
   RHSUSES P1 P2 P3 P4 P5 P8 P9 M20 -M20 M30 -M30 M40 M50 -M50 M55 -M55
 HASREPR
   LHSUSES B8
   NESTEDL B5
   RHSUSES B6 B8 -B8
 HASSQUARE
   LHSUSES C15 C17
   RHSUSES P45 -C15 -C17
 HASSQUARED
   LHSUSES C50 C52
   RHSUSES P65 -C50 -C52
 HASUOPCHUNK
   LHSUSES V10
   RHSUSES C15 C50
 HASVERB
   LHSUSES M20 M30 M40 M50 M55
   RHSUSES P30 -M20 -M30 -M40 -M50 -M55
 HIGHPREC
   LHSUSES S20 S30 S60 P20 P23 P27 P28 P30 P35 P40 P45 P50 P65 P70 P75
   RHSUSES S10 S20 -S20 S40 -S60 S65 P10 P15 P20 -P20 P27 -P27 -P30 -P35 -P40
   -P45 -P50 -P65 -P70 -P75
 IFDELETED
   LHSUSES T26
   NESTEDL T26D
   RHSUSES T29
 INCHUNK
   LHSUSES M20 M30 M50 M55 M60 M62 R2 R4 R6 R7 R8 V15 V21
   NESTEDL M40 M50 M55 V20
   RHSUSES S13 S15 S60 -M20 -M30 M60 M62 C15 C17 C20 -C20 C22 -C22 C50 C52 R2
   -R2 R4 -R4 R6 -R6 R7 R8 -R8 R9 -R9
 ISANSUNIT
   LHSUSES B3
   RHSUSES F40 B3 -B3
```

```
 ISCHUNK
   LHSUSES P10 P15
   RHSUSES S40 P10 M40 C15 C50 R4 R8 R9
 ISDELIM
   LHSUSES S13 S40
   RHSUSES D91
 ISEQN
   RHSUSES C90
 ISEV
   LHSUSES F80 B2
   NESTEDL B2
   RHSUSES S60 F15 F75 -B2
 ISIS
   LHSUSES P7
   RHSUSES D96
 ISOP0
   LHSUSES P8 P9
   RHSUSES D13 D14 D15 D17 D18 D19
 ISOP1
   LHSUSES P4 P5 M65 M75
   RHSUSES D5 D7 D9 D11 M60 M62 -M75
 ISOP2
   LHSUSES P3
   RHSUSES D1 D3
 ISPERSON
   LHSUSES A61 A62
   RHSUSES D41 D44 D47 D50 D53 A61
 ISPLURAL
   LHSUSES M60 V34 B3
   RHSUSES D61 D63 D65 D67 D69 D71 D72 D73 D75 D77 D78 D79
 ISPOSSPRON
   LHSUSES A81
   RHSUSES D57
 ISPRON
   LHSUSES A71
   RHSUSES D55
 ISQWORD
   LHSUSES S60
   RHSUSES D81 D83 D85 D87
 ISRELEXPR
   LHSUSES C80 V15
   NESTEDL C85 V21
   RHSUSES C80 -C80 C85 -V15
 ISSCANCHUNK
   LHSUSES S13 S20 S25 S30 S40 S60 T21 T22 T23 T24 T31 M62 A38
   RHSUSES S10 S40 -S40 -S60 S65
 ISSCANEV
   LHSUSES S15 S35 F5 F25 F40 F45 F50 F60
   RHSUSES S60 -S65
 ISSINGULAR
   LHSUSES M62
   RHSUSES D67S D71S
 ISUOPDUM
   LHSUSES V10
   NESTEDL V5
   RHSUSES C15 C50
 ISVARCHUNK
   LHSUSES V5 V10 V15 V20 V21 V23 V25 B5
   NESTEDL V20 V55 B5
   RHSUSES P75 -V10 F15 F20 F35 F70 F75
 ISVERB
   LHSUSES P1 M20 M30 M50 M55 -M60
   RHSUSES D21 D24 D27 D30 -M20 -M30 -M50 -M55
 LABELE
   LHSUSES C70 C75 C78 C90
   RHSUSES P75 C70 C75 C78
 LABELU
   LHSUSES P75 M20 M30 M50 M55 C15 C50 C60 C70 C75 C78
   RHSUSES S40 -P75 M20 -M20 M30 -M30 M50 -M50 M55 -M55 C15 C50 C60 -C70 -C75
   -C78
 LRENAM
   LHSUSES R2 R4
   RHSUSES C60 R2 -R2 -R4
 MODLEN
   LHSUSES I5
   RHSUSES T1 T2 T3 T4 T5 T6 T8 T9 T9F T10 T11 T12 T15 T16 T17 T18 T19 T20 T21
   T23 T25 T26D T29 T30 M60 M62 A11 A12 A15 A17 A19 A20 A24 A26 A28 A31 A32 A34
   A35 A41 A59 A61 A75 A85 -I5 I7
 MODLENC
   LHSUSES I7
   RHSUSES D91 D96 -I7 I11
 MXCPRIOR
   LHSUSES S40 S65 C60 F80
   RHSUSES S10 S40 -S40 S65 -S65 C60 -C60 F80 -F80
  NEWDVAR
   LHSUSES I31
   RHSUSES V60 -I31
  NEWEQN
   LHSUSES I21
   RHSUSES M10 -I21
  NEWEV
   LHSUSES I41 I43
   RHSUSES S60 F15 F75 -I41 -I43
  NEWOP
   LHSUSES I11
   RHSUSES D1 D3 D5 D7 D9 D13 D14 D15 D17 D18 D19 M60 M62 M65 -I11
  NEWPLOP
   LHSUSES I15 I17
   RHSUSES C60 -I15 -I17
  NEWPLVAR
   LHSUSES I33
   RHSUSES V50 V60 -I33
  NEWREFEXPR
   LHSUSES C80 C85
   RHSUSES C75 C78 -C80 -C85
  NEWREFOP
   LHSUSES I13
   RHSUSES C2 C5 C8 -I13
  NEWSIZE
   LHSUSES I3
   RHSUSES I1 -I3 I5 I13 I15 I21 I31 I33 I41 I43
  NUMVARCHUNKS
   LHSUSES V5 V50 V60
   RHSUSES V5 -V5 V60 -V50 I1
  PLACOPLIST
   RHSUSES I15 I17
  PRECSCAN
   LHSUSES P20 P23 P26 P27 P28 P29
   RHSUSES P10 P15 P20 -P20 P23 -P23 P26 -P26 -P27 -P28 -P29
  PRECSCAND
   LHSUSES -P10 -P15 P30 P35 P40 P45 P50 P65 P70 P75
   NESTEDL -P10 -P15
   RHSUSES S40 P27 P28 P29 M40 C15 -C50
  PROBEQNS
   LHSUSES I3 I5 I21 I41
   RHSUSES I1 I5 -I5 I21 -I21 I41 -I41
  PROBEVS
   LHSUSES I3 I5 I31 I33 I41 I43
   RHSUSES I1 I5 -I5 I31 -I31 I41 -I41 I43 -I43
  PROBLEM
   LHSUSES S70 I1
  PROBOPS
   LHSUSES I3 I5 I11 I13 I15 I41
   RHSUSES I1 I5 -I5 I11 -I11 I13 -I13 I15 -I15 I41 -I41
  PROBVARS
   LHSUSES I3 I5 I11 I13 I21 I31 I33 I41
   RHSUSES I1 I5 -I5 I11 -I11 I13 I13 I21 -I21 I31 -I31 I33 -I33 I41 -I41
  RRENAME
   LHSUSES R6 R7 R8 R9
   RHSUSES M20 M30 M50 M55 C60 R6 R6 R7 -R7 -R8 R9
  RTAMDIFERGOING
   LHSUSES F70 F75
   RHSUSES F60 -F70 F75 -F75
  RTAMDQMGOING
   LHSUSES F15 F20
   RHSUSES F5 F15 -F15 -F20
  RTDOESGOING
   LHSUSES F52
   RHSUSES F50 -F52
  RTDOGOING
   LHSUSES F48
   RHSUSES F45 -F48
  RTHAVEGOING
   LHSUSES F55
   RHSUSES F52 -F55
  RTQMGOING
   LHSUSES F35
   RHSUSES F25 -F35 F40 F48 F55
  SPACESIZEN
   LHSUSES I3 I11 I13 I15 I17
   NESTEDL I1
   RHSUSES I1 I3 -I3
  SPACESIZES
   RHSUSES I1 I3
  STRLENGTH
   LHSUSES I1 I5
   RHSUSES I5 -I5
  TAMDIFF
   LHSUSES -T22 T24
   RHSUSES T23 -T24
  TANDSUM
   LHSUSES T22
   RHSUSES T21
  TOYIS
   LHSUSES T32
   RHSUSES T31 -T32
  TEASCAN
   LHSUSES A11 A12 A15 A17 A18 A19 A20 A24 A26 A28 A31 A32 A34 A35 A38 A50
   RHSUSES S16 -S18 -A11 -A12 -A15 -A17 -A18 A19 -A20 -A24 -A26 -A28 A31 -A32
   -A34 -A35 -A38 A41 -A50
  TEASCANFIN
   LHSUSES S18
   RHSUSES S16 -S18 -T50 -T51
  TFOUT
   LHSUSES T50 T51
   RHSUSES T13 T16 T21 T23 T26D T29 T30 -T50 -T51 T52 A11 A12 A24 A26 A28 A77
  TFOUTDELAY
   LHSUSES A77
   RHSUSES A50 A75 -A77 A85
  TFOUTLEN
   LHSUSES T52
   RHSUSES -T52 A15 A17
  TFSCAN
   LHSUSES T1 T2 T2S T3 T4 T5 T6 T7 T8 T9 T9S T10 T11 T12 T13 T14 T15 T16 T17
   T18 T19 T20 T21 T22 T23 T24 T25 T26 T26D T27 T28 T29 T30 T31 T32
   RHSUSES S10 S13 S15 -S16 -S17 S40 S65 -T1 -T2 -T2S -T3 -T4 -T5 -T6 T7 -T8
   -T9 -T9S -T10 -T11 -T12 -T13 -T14 -T15 -T16 -T17 -T18 T19 -T20 -T21 -T22
   -T23 -T24 -T25 -T26 -T26D -T27 -T28 -T29 -T30 -T31 -T32 T50 T51 A18 A19
  TFSCANFIN
   LHSUSES S16 S17
   RHSUSES S10 S13 S15 -S16 S17 S40 S65 T50 -T50 T51 -T51 A18 A19
  TGSCAN
   LHSUSES D1 D3 D5 D7 D9 D11 D13 D14 D15 D17 D18 D19 D21 D24 D27 D30 D41 D44
   D47 D50 D53 D55 D57 D61 D63 D65 D67 D67S D69 D71 D71S D72 D73 D75 D77 D78 D79
   D81 D83 D85 D87 D91 D96
   RHSUSES -S13 -S15 S17 S18 -D1 -D3 -D5 -D7 -D9 -D11 -D13 -D14 -D15 -D17 -D18
   -D19 -D21 -D24 D27 -D30 -D41 -D44 -D47 -D50 -D53 -D55 -D57 -D61 -D63 -D65
   -D67 -D67S -D69 -D71 -D71S -D72 -D73 D75 -D77 -D78 D79 -D81 -D83 -D85 -D87
   -D91 -D96
  TGSCANFIN
   LHSUSES S10 S20 S25 S30 S35
   RHSUSES -S10 S17 S18 -S20 -S25 -S30 -S35 -A75 A85
  TGSCANFIN2
   LHSUSES S13 S15 S40
   RHSUSES -S13 -S15 S20 S25 S30 S35 -S40 -S65
  THEIRCOLL
   LHSUSES A52 A53 A56 A57
   RHSUSES A51 A52 -A52 -A56 -A57
  THEIRCOLLD
   LHSUSES -A51 A56 A57
   NESTEDL -A51 -A59
   RHSUSES A53
  THEIRREF
   LHSUSES A51 A59
   RHSUSES A50 A51 -A59
  THEIRREFL
   LHSUSES A56 A59
   NESTEDL A57
   RHSUSES A56 -A56 A57 -A59
  THISTESTED
   LHSUSES V23 V24
   RHSUSES V20 V21 -V23 -V24
  UNTESTED
   LHSUSES V15 V20 V21
   RHSUSES V5 -V15 -V20 -V21
  URENAME
   LHSUSES C20 C22
   RHSUSES C15 C20 -C20 -C22 C50
  VARCHCOUNT
   LHSUSES V55 V85
   RHSUSES V25 -V55 -V85
  VARCLEANUP
   LHSUSES V65 V80 V85 V90
   RHSUSES V15 V50 V60
  WCOLLECT
   LHSUSES B8
   RHSUSES B6 B8 -B8
```

Appendix D. SUMMARY OF CONTROL FLOW

```
(TEST2 TRACE FOR CONTROL FLOW SUMMARY)

          S T D P M C R V F

S10-1     •                   S...........11
D3-1          •               D.1
P3-1            •             P.1
S20-1     •                   S....5
T2-1        •                 T.1
S17-6     •                   S.1
D96-1         •               D.1
P2-1            •             P.1
S20-2     •                   S...........11
T28-1       •                 T.1
S17-10    •                   S.1
D91-1         •               D.1
S25-8     •                   S..2
P35-1           •             P.1
M10-1             •           M.1
C60-1               •         C.1
R6-1                  •       R........8
P10-1           •             P.......7
C10-1               •         C..2
R8-2                  •       R....4
P10-2           •             P.....5
V5-1                    •     V.....5
P10-3           •             P...3
V5-2                    •     V........8
C20-1               •         C.1
P15-1           •             P.....5
V5-3                    •     V.............13
C25-1               •         C...3
T12-1       •                 T.1
S17-11    •                   S....4
D5-1          •               D.1
P5-1            •             P.1
S20-3     •                   S............17
D96-2         •               D.1
P2-2            •             P.1
S20-4     •                   S......6
D5-2          •               D.1
P5-2            •             P.1
S30-1     •                   S..2
T11-1       •                 T.1
S17-19    •                   S....4
D11-1         •               D.1
M65-1             •           M.1
P5-3            •             P.1
S30-2     •                   S...........11
T20-2       •                 T.1
S17-24    •                   S.1
D91-2         •               D.1
S25-10    •                   S..2
P35-2           •             P.1
M10-2             •           M.1
C60-3               •         C.1
R6-3                  •       R............17
P10-4           •             P.......7
C25-1               •         C..2
R6-9                  •       R. ..4
P10-5           •             P...3
V5-4                    •     V................16
P10-6           •             P.....5
V5-5                    •     V....................20
C20-2               •         C.1
P15-2           •             P.........9
C25-2               •         C..2
R6-11                 •       R......6
P10-7           •             P...3
V5-6                    •     V.....................21
P15-3           •             P.......7
C25-3               •         C..2
R6-15                 •       R...4
P10-8           •             P...3
V5-7                    •     V.........................25
P15-4           •             P....5
V5-8                    •     V..........................27
C20-3               •         C.....5
S17-25    •                   S.1
D81-1         •               D.1
```

```
S60-1     •                   S.1
F80-1                     •   F.1
S35-1     •                   S.1
F5-1                      •   F.1
S15-1     •                   S..............15
F15-1                     •   F.1
V5-9                    •     V............................29
F80-2                     •   F.1
S15-6     •                   S.........10
T22-1       •                 T.1
S17-34    •                   S..2
F20-1                     •   F.1
V5-10                   •     V...............................31
S20-1     •                   S....4

TOTAL FIRINGS OF EACH TYPE, SCALE FACTOR 3
S ...................................107
T ..6
D ...9
P ........................70
M .3
C ......20
R ............38
V ..............................................................195
F .5
B ......19
I ...........................82
```

## Appendix E. RESULTS FOR 27 TESTS

```
TEST1
((A PLUS B IS 5 . B IS 3 . FIND A .))
ISEQN (C-1 (EQUAL (PLUS VAR-1 VAR-2) VAR-3)) (C-2 (EQUAL VAR-2 VAR-4))
HASREPR (VAR-1 (A)) (VAR-2 (B)) (VAR-3 (5)) (VAR-4 (3))
EVLIST (PB-1 ((VAR-1)))
EQVARCHUNK (C-3 CL-2) (CL-3 CR-2)
HASEXPR (C-1 (EQUAL (PLUS VAR-1 VAR-2) VAR-3)) (C-2 (EQUAL VAR-2 VAR-4))
  (C-3 VAR-1) (CL-1 (PLUS VAR-1 VAR-2)) (CL-2 VAR-1) (CL-3 VAR-2) (CR-1 VAR-3)
  (CR-2 VAR-2) (CR-3 VAR-4)

RUN TIME 1 MIN, 23.2 SEC

EXAM    TRY    FIRE   WMACT  E/F   E/T   T/F
1835    450    235    937    7.81  4.08  1.91
0.0453  0.185  0.354  0.0888 SEC AVG

557 INSERTS 380 DELETES 26 WARNINGS 31 NEW OBJECTS
MAX :SMPX LENGTH 135
CORE (FREE.FULL): (8219 . 1333) USED (2985 . 255)

FIRED 69 OUT OF 260 PRODS

TEST2
  ((A FIRST NUMBER PLUS 6 IS EQUAL TO A SECOND NUMBER . TWICE THE FIRST NUMBER
      IS THREE TIMES ONE HALF OF THE SECOND NUMBER . WHAT ARE THE FIRST NUMBER
      AND THE SECOND NUMBER ?))
ISEQN (C-1 (EQUAL (PLUS VAR-1 VAR-2) VAR-3))
  (C-2 (EQUAL (TIMES VAR-4 VAR-1) (TIMES VAR-5 (TIMES VAR-6 VAR-3))))
HASREPR (VAR-1 (A FIRST NUMBER)) (VAR-2 (6)) (VAR-3 (A SECOND NUMBER))
  (VAR-4 (2)) (VAR-5 (THREE)) (VAR-6 (0.5))
EVLIST (PB-1 ((VAR-1 VAR-3)))
EQVARCHUNK (C-3 CL-2) (C-4 CR-1) (CR-4 CL-2) (CR-6 CR-1)
HASEXPR (C-1 (EQUAL (PLUS VAR-1 VAR-2) VAR-3))
  (C-2 (EQUAL (TIMES VAR-4 VAR-1) (TIMES VAR-5 (TIMES VAR-6 VAR-3))))
  (C-3 VAR-1) (C-4 VAR-3) (CL-1 (PLUS VAR-1 VAR-2)) (CL-2 VAR-1)
  (CL-3 (TIMES VAR-4 VAR-1)) (CL-4 VAR-4) (CL-5 VAR-5) (CL-6 VAR-6) (CR-1 VAR-3)
  (CR-2 VAR-2) (CR-3 (TIMES VAR-5 (TIMES VAR-6 VAR-3))) (CR-4 VAR-1)
  (CR-5 (TIMES VAR-6 VAR-3)) (CR-6 VAR-3)

RUN TIME 4 MIN, 38.3 SEC

EXAM    TRY    FIRE   WMACT  E/F   E/T   T/F
4445    1229   555    2115   8.01  3.62  2.21
0.0626  0.226  0.502  0.132  SEC AVG

1218 INSERTS 807 DELETES 64 WARNINGS 64 NEW OBJECTS
MAX :SMPX LENGTH 125
CORE (FREE.FULL): (4706 . 837) USED (6498 . 781)

FIRED 87 OUT OF 260 PRODS

TEST3
  ((A NUMBER IS MULTIPLIED BY 6 . THIS PRODUCT IS INCREASED BY 44 . THIS RESULT
      IS 68 . FIND THE NUMBER .))
ISEQN (C-3 (EQUAL (PLUS (TIMES VAR-1 VAR-2) VAR-3) VAR-4))
HASREPR (VAR-1 (A NUMBER)) (VAR-2 (6)) (VAR-3 (44)) (VAR-4 (68))
  ((TIMES VAR-1 VAR-2) (THIS PRODUCT))
  ((PLUS (TIMES VAR-1 VAR-2) VAR-3) (THIS RESULT))
EVLIST (PB-1 ((VAR-1)))
EQVARCHUNK (C-4 CL-1)
HASEXPR (C-1 (TIMES VAR-1 VAR-2)) (C-2 (PLUS (TIMES VAR-1 VAR-2) VAR-3))
  (C-3 (EQUAL (PLUS (TIMES VAR-1 VAR-2) VAR-3) VAR-4)) (C-4 VAR-1) (CL-1 VAR-1)
  (CL-2 (TIMES VAR-1 VAR-2)) (CL-3 (PLUS (TIMES VAR-1 VAR-2) VAR-3))
  (CR-1 VAR-2) (CR-2 VAR-3) (CR-3 VAR-4)

RUN TIME 1 MIN, 57.5 SEC

EXAM    TRY    FIRE   WMACT  E/F   E/T   T/F
2919    604    289    1189   10.1  4.83  2.09
0.0403  0.195  0.407  0.0988 SEC AVG

710 INSERTS 479 DELETES 31 WARNINGS 43 NEW OBJECTS
MAX :SMPX LENGTH 105
CORE (FREE.FULL): (7323 . 1230) USED (3881 . 384)

FIRED 67 OUT OF 260 PRODS

TEST4
  ((IF THE NUMBER OF CUSTOMERS TOM GETS IS TWICE THE SQUARE OF 20 PER CENT OF
      THE NUMBER OF ADVERTISEMENTS HE RUNS , AND THE NUMBER OF ADVERTISEMENTS
      HE RUNS IS 45 , WHAT IS THE NUMBER OF CUSTOMERS TOM GETS ?))
ISEQN (C-1 (EQUAL VAR-1 (TIMES VAR-4 (EXPT (TIMES VAR-2 VAR-3) VAR-4))))
  (C-2 (EQUAL VAR-3 VAR-5))
HASREPR (VAR-1 (THE NUMBER OF CUSTOMERS TOM GETS)) (VAR-2 (0.19999999))
  (VAR-3 (THE NUMBER OF ADVERTISEMENTS HE RUNS)) (VAR-4 (2)) (VAR-5 (45))
EVLIST (PB-1 ((VAR-1)))
EQVARCHUNK (C-3 CL-1) (CL-4 CR-2) (CL-5 CR-3)
HASEXPR (C-1 (EQUAL VAR-1 (TIMES VAR-4 (EXPT (TIMES VAR-2 VAR-3) VAR-4))))
  (C-2 (EQUAL VAR-3 VAR-5)) (C-3 VAR-1) (CL-1 VAR-1) (CL-2 (TIMES VAR-2 VAR-3))
  (CL-3 VAR-2) (CL-4 VAR-4) (CL-5 VAR-3)
  (CR-1 (TIMES VAR-4 (EXPT (TIMES VAR-2 VAR-3) VAR-4))) (CR-2 VAR-4)
  (CR-3 VAR-5) (CR-4 (EXPT (TIMES VAR-2 VAR-3) VAR-4)) (CR-5 VAR-4)
  (CU-1 (EXPT (TIMES VAR-2 VAR-3) VAR-4))

RUN TIME 5 MIN, 2.90 SEC

EXAM    TRY    FIRE   WMACT  E/F   E/T   T/F
4740    1484   557    2192   8.51  3.19  2.66
0.0639  0.204  0.544  0.138  SEC AVG

1262 INSERTS 930 DELETES 58 WARNINGS 70 NEW OBJECTS
MAX :SMPX LENGTH 153
CORE (FREE.FULL): (3613 . 756) USED (6422 . 776)

FIRED 96 OUT OF 260 PRODS

TEST5
  ((THE SUM OF LOIS SHARE OF SOME MONEY AND BOB S SHARE IS $ 4.5 . LOIS SHARE IS
      TWICE BOB S . FIND BOB S AND LOIS SHARE .))
ISEQN (C-1 (EQUAL (PLUSS VAR-1 VAR-2) (TIMES VAR-3 VAR-4)))
  (C-2 (EQUAL VAR-1 (TIMES VAR-5 VAR-2)))
HASREPR (VAR-1 (LOIS SHARE OF SOME MONEY)) (VAR-2 (BOB S SHARE)) (VAR-3 (4.5))
  (VAR-4 (DOLLAR)) (VAR-5 (2))
EVLIST (PB-1 ((VAR-2 VAR-1)))
EQVARCHUNK (C-3 CR-2) (C-4 CL-2) (CL-4 CL-2) (CR-5 CR-2)
HASEXPR (C-1 (EQUAL (PLUSS VAR-1 VAR-2) (TIMES VAR-3 VAR-4)))
  (C-2 (EQUAL VAR-1 (TIMES VAR-5 VAR-2))) (C-3 VAR-2) (C-4 VAR-1)
  (CL-1 (PLUSS VAR-1 VAR-2)) (CL-2 VAR-1) (CL-3 VAR-3) (CL-4 VAR-1) (CL-5 VAR-5)
  (CR-1 (TIMES VAR-3 VAR-4)) (CR-2 VAR-2) (CR-3 VAR-4)
  (CR-4 (TIMES VAR-5 VAR-2)) (CR-5 VAR-2)

RUN TIME 3 MIN, 50.1 SEC

EXAM    TRY    FIRE   WMACT  E/F   E/T   T/F
3691    990    468    1831   7.89  3.73  2.12
0.0623  0.232  0.492  0.126  SEC AVG

1057 INSERTS 774 DELETES 56 WARNINGS 57 NEW OBJECTS
MAX :SMPX LENGTH 128
CORE (FREE.FULL): (4475 . 899) USED (5560 . 633)

FIRED 89 OUT OF 260 PRODS

TEST6
  ((MARY IS TWICE AS OLD AS ANN WAS WHEN MARY WAS AS OLD AS ANN IS NOW . IF MARY
      IS 24 YEARS OLD . HOW OLD IS ANN ?))
ISEQN (C-1 (EQUAL VAR-1 (TIMES VAR-2 (MINUSS VAR-3 VAR-4))))
  (C-2 (EQUAL (MINUSS VAR-1 VAR-4) VAR-3)) (C-3 (EQUAL VAR-1 VAR-5))
HASREPR (VAR-1 (MARY S AGE)) (VAR-2 (2)) (VAR-3 (ANN S AGE)) (VAR-4 (G5506))
  (VAR-5 (24))
EVLIST (PB-1 ((VAR-3)))
EQVARCHUNK (C-4 CL-3) (CL-5 CL-1) (CL-6 CL-1) (CR-4 CL-3) (CR-5 CR-3)
HASEXPR (C-1 (EQUAL VAR-1 (TIMES VAR-2 (MINUSS VAR-3 VAR-4))))
  (C-2 (EQUAL (MINUSS VAR-1 VAR-4) VAR-3)) (C-3 (EQUAL VAR-1 VAR-5)) (C-4 VAR-3)
  (CL-1 VAR-1) (CL-2 VAR-2) (CL-3 VAR-3) (CL-4 (MINUSS VAR-1 VAR-4))
  (CL-5 VAR-1) (CL-6 VAR-1) (CR-1 (TIMES VAR-2 (MINUSS VAR-3 VAR-4)))
  (CR-2 (MINUSS VAR-3 VAR-4)) (CR-3 VAR-4) (CR-4 VAR-3) (CR-5 VAR-4)
  (CR-6 VAR-5)

RUN TIME 7 MIN, 25.3 SEC

EXAM    TRY    FIRE   WMACT  E/F   E/T   T/F
5556    1677   686    2811   8.10  3.31  2.44
0.0801  0.266  0.649  0.158  SEC AVG

1590 INSERTS 1221 DELETES 106 WARNINGS 83 NEW OBJECTS
MAX :SMPX LENGTH 100
CORE (FREE.FULL): (1948 . 568) USED (8081 . 864)

FIRED 109 OUT OF 260 PRODS

TEST7
  ((THE SUM OF THE PERIMETER OF A RECTANGLE AND THE PERIMETER OF A TRIANGLE IS
      24 INCHES . IF THE PERIMETER OF THE RECTANGLE IS TWICE THE PERIMETER OF
```

E.

```
     THE TRIANGLE , WHAT IS THE PERIMETER OF THE TRIANGLE ?))
ISEQN (C-1 (EQUAL (PLUSS VAR-1 VAR-2) (TIMES VAR-3 VAR-4)))
 (C-2 (EQUAL VAR-1 (TIMES VAR-5 VAR-2)))
HASREFR (VAR-1 (THE PERIMETER OF A RECTANGLE))
 (VAR-2 (THE PERIMETER OF A TRIANGLE)) (VAR-3 (24)) (VAR-4 (INCH)) (VAR-5 (2))
EVLIST (PH-1 ((VAR-2)))
EQVARCHUNK ((C-3 CR-2) (CL-4 CL-2) (CR-5 CR-2)
HASEXPR (C-1 (EQUAL (PLUSS VAR-1 VAR-2) (TIMES VAR-3 VAR-4)))
 (C-2 (EQUAL VAR-1 (TIMES VAR-5 VAR-2))) (C-3 VAR-2) (CL-1 (PLUSS VAR-1 VAR-2))
 (CL-2 VAR-1) (CL-3 VAR-3) (CL-4 VAR-1) (CL-5 VAR-5) (CR-1 (TIMES VAR-3 VAR-4))
 (CR-2 VAR-2) (CR-3 VAR-4) (CR-4 (TIMES VAR-5 VAR-2)) (CR-5 VAR-2)

RUN TIME 4 MIN 31.5 SEC

EXAM    TRY    FIRE   WMACT  E/F   E/T   T/F
4584    1312   528    2069   8.68  3.49  2.48
0.0597  0.207  0.510  0.131  SEC AVG

1192 INSERTS 827 DELETES 51 WARNINGS 66 NEW OBJECTS
MAX SMPX LENGTH 137
CORE (FREE.FULL): (4161 . 853) USED (6178 . 739)

FIRED 89 OUT OF 260 PRODS

TEST8
 ((THE PRICE OF A RADIO IS 69.699997 DOLLARS . IF THIS PRICE IS 15 PER CENT
     LESS THAN THE MARKED PRICE , FIND THE MARKED PRICE .))
ISEQN (C-1 (EQUAL VAR-1 (TIMES VAR-2 VAR-3)))
 (C-2 (EQUAL VAR-1 (TIMES VAR-4 VAR-5)))
HASREFR (VAR-1 (THE PRICE OF A RADIO)) (VAR-2 (69.699997)) (VAR-3 (DOLLAR))
 (VAR-4 (0.85000000)) (VAR-5 (THE MARKEDPRICE))
EVLIST (PH-1 ((VAR-5)))
EQVARCHUNK ((C-3 CR-4)
HASEXPR (C-1 (EQUAL VAR-1 (TIMES VAR-2 VAR-3)))
 (C-2 (EQUAL VAR-1 (TIMES VAR-4 VAR-5))) (C-3 VAR-5) (CL-1 VAR-1) (CL-2 VAR-2)
 (CL-3 VAR-1) (CL-4 VAR-4) (CR-1 (TIMES VAR-2 VAR-3)) (CR-2 VAR-3)
 (CR-3 (TIMES VAR-4 VAR-5)) (CR-4 VAR-5)

RUN TIME 3 MIN 13.6 SEC

EXAM    TRY    FIRE   WMACT  E/F   E/T   T/F
3003    840    353    1454   8.51  3.58  2.38
0.0645  0.229  0.548  0.133  SEC AVG

844 INSERTS 606 DELETES 36 WARNINGS 50 NEW OBJECTS
MAX SMPX LENGTH 131
CORE (FREE.FULL): (7234 . 1126) USED (4380 . 465)

FIRED 80 OUT OF 260 PRODS

TEST9
 ((BILL IS ONE HALF OF HIS FATHER S AGE 4 YEARS AGO . IN 20 YEARS HE WILL BE 2
     YEARS OLDER THAN HIS FATHER IS NOW . HOW OLD ARE BILL AND HIS FATHER ?))
ISEQN (C-1 (EQUAL VAR-1 (TIMES VAR-2 (MINUSS VAR-3 VAR-4))))
 (C-2 (EQUAL (PLUSS VAR-1 VAR-5) (PLUS VAR-6 VAR-3)))
HASREFR (VAR-1 (BILL S AGE)) (VAR-2 (0.5)) (VAR-3 (BILL S FATHER S AGE))
 (VAR-4 (4)) (VAR-5 (20)) (VAR-6 (2))
EVLIST (PH-1 (VAR-1 VAR-3)))
EQVARCHUNK ((C-3 CL-1) (C-4 CL-3) (CL-5 CL-1) (CR-6 CL-3)
HASEXPR (C-1 (EQUAL VAR-1 (TIMES VAR-2 (MINUSS VAR-3 VAR-4))))
 (C-2 (EQUAL (PLUSS VAR-1 VAR-5) (PLUS VAR-6 VAR-3))) (C-3 VAR-1) (C-4 VAR-3)
 (CL-1 VAR-1) (CL-2 VAR-2) (CL-3 VAR-3) (CL-4 (PLUSS VAR-1 VAR-5)) (CL-5 VAR-1)
 (CL-6 VAR-6) (CR-1 (TIMES VAR-2 (MINUSS VAR-3 VAR-4)))
 (CR-2 (MINUSS VAR-3 VAR-4)) (CR-3 VAR-4) (CR-4 (PLUS VAR-6 VAR-3))
 (CR-5 VAR-5) (CR-6 VAR-3)

RUN TIME 10 MIN 00.6 SEC

EXAM    TRY    FIRE   WMACT  E/F   E/T   T/F
6436    2028   777    3029   8.28  3.17  2.61
0.0935  0.296  0.824  0.212  SEC AVG

1704 INSERTS 1325 DELETES 102 WARNINGS 86 NEW OBJECTS
MAX SMPX LENGTH 137
CORE (FREE.FULL): (1698 . 467) USED (8901 . 1125)

FIRED 123 OUT OF 260 PRODS

TEST10
 ((BILL S FATHER S UNCLE IS TWICE AS OLD AS BILL S FATHER . 2 YEARS FROM NOW
     BILL S FATHER WILL BE 3 TIMES AS OLD AS BILL . THE SUM OF THEIR AGES IS
     92 . FIND BILL S AGE .))
ISEQN (C-1 (EQUAL VAR-1 (TIMES VAR-2 VAR-3)))
```

```
 (C-2 (EQUAL (PLUSS VAR-3 VAR-2) (TIMES VAR-4 (PLUSS VAR-5 VAR-2))))
 (C-3 (EQUAL (PLUSS VAR-1 (PLUSS VAR-3 VAR-5)) VAR-6))
HASREFR (VAR-1 (BILL S FATHER S UNCLE S AGE)) (VAR-2 (2))
 (VAR-3 (BILL S FATHER S AGE)) (VAR-4 (3)) (VAR-5 (BILL S AGE)) (VAR-6 (92))
EVLIST (PH-1 ((VAR-5)))
EQVARCHUNK ((C-4 CL-6) ((CL-4 CR-2) (CL-8 CL-1) (CL-9 CR-2) (CR-4 CL-2)
 ((CR-6 CL-2) (CR-9 CL-6)
HASEXPR (C-1 (EQUAL VAR-1 (TIMES VAR-2 VAR-3)))
 (C-2 (EQUAL (PLUSS VAR-3 VAR-2) (TIMES VAR-4 (PLUSS VAR-5 VAR-2))))
 (C-3 (EQUAL (PLUSS VAR-1 (PLUSS VAR-3 VAR-5)) VAR-6)) (C-4 VAR-5) (CL-1 VAR-1)
 (CL-2 VAR-2) (CL-3 (PLUSS VAR-3 VAR-2)) (CL-4 VAR-3) (CL-5 VAR-4) (CL-6 VAR-5)
 (CL-7 (PLUSS VAR-1 (PLUSS VAR-3 VAR-5))) (CL-8 VAR-1) (CL-9 VAR-3)
 (CR-1 (TIMES VAR-2 VAR-3)) (CR-2 VAR-3)
 (CR-3 (TIMES VAR-4 (PLUSS VAR-5 VAR-2))) (CR-4 VAR-2)
 (CR-5 (PLUSS VAR-5 VAR-2)) (CR-6 VAR-2) (CR-7 VAR-6)
 (CR-8 (PLUSS VAR-3 VAR-5)) (CR-9 VAR-5)

RUN TIME 10 MIN 28.0 SEC

EXAM    TRY    FIRE   WMACT  E/F   E/T   T/F
8226    2807   1174   4333   7.01  2.93  2.39
0.0763  0.224  0.535  0.145  SEC AVG

2424 INSERTS 1909 DELETES 180 WARNINGS 113 NEW OBJECTS
MAX SMPX LENGTH 135
CORE (FREE.FULL): (5889 . 765) USED (4876 . 761)

FIRED 113 OUT OF 259 PRODS

TEST11
 ((TOM HAS TWICE AS MANY FISH AS MARY HAS GUPPIES . IF MARY HAS 3 GUPPIES , HOW
     MANY FISH DOES TOM HAVE ?))
ISEQN ((CN-1 ((EQUAL (TIMES VAR-1 VAR-2) VAR-3)) (CN-2 (EQUAL VAR-3 VAR-4)))
HASREFR (VAR-1 (2)) (VAR-2 (THE NUMBER OF FISH TOM HAS))
 (VAR-3 (THE NUMBER OF GUPPIES MARY HAS)) (VAR-4 (3))
EVLIST (PH-1 ((VAR-2)))
EQVARCHUNK ((C-3 CR-2) (CL-3 CR-1)
HASEXPR ((C-3 VAR-2) (CL-1 (TIMES VAR-1 VAR-2)) (CL-2 VAR-1) (CL-3 VAR-3)
 (CN-1 (EQUAL (TIMES VAR-1 VAR-2) VAR-3)) (CN-2 (EQUAL VAR-3 VAR-4))
 (CR-1 VAR-3) (CR-2 VAR-2) (CR-3 VAR-4)

RUN TIME 4 MIN 1.92 SEC

EXAM    TRY    FIRE   WMACT  E/F   E/T   T/F
3341    1081   406    1627   8.23  3.09  2.66
0.0724  0.224  0.596  0.149  SEC AVG

986 INSERTS 681 DELETES 41 WARNINGS 61 NEW OBJECTS
MAX SMPX LENGTH 131
CORE (FREE.FULL): (5570 . 982) USED (4795 . 544)

FIRED 90 OUT OF 260 PRODS

TEST12
 ((IF 1 SPAN EQUALS 9 INCHES , AND 1 FATHOM EQUALS 6 FEET , HOW MANY SPANS
     EQUALS 1 FATHOM ?))
ISEQN (C-1 (EQUAL (TIMES VAR-1 VAR-2) (TIMES VAR-3 VAR-4)))
 (C-2 (EQUAL (TIMES VAR-1 VAR-5) (TIMES VAR-6 VAR-7)))
HASREFR (VAR-1 (1)) (VAR-2 (SPAN)) (VAR-3 (9)) (VAR-4 (INCH)) (VAR-5 (FATHOM))
 (VAR-6 (6)) (VAR-7 (FOOT))
EVLIST (PH-1 ((VAR-5)))
ISANSUNIT (SPAN)
EQVARCHUNK ((C-3 CR-5) ((CL-5 CL-7)
HASEXPR (C-1 (EQUAL (TIMES VAR-1 VAR-2) (TIMES VAR-3 VAR-4)))
 (C-2 (EQUAL (TIMES VAR-1 VAR-5) (TIMES VAR-6 VAR-7))) (C-3 VAR-5)
 (CL-1 (TIMES VAR-1 VAR-2)) (CL-2 VAR-1) (CL-3 VAR-3)
 (CL-4 (TIMES VAR-1 VAR-5)) (CL-5 VAR-1) (CL-6 VAR-6)
 (CR-1 (TIMES VAR-3 VAR-4)) (CR-2 VAR-2) (CR-3 VAR-4)
 (CR-4 (TIMES VAR-6 VAR-7)) (CR-5 VAR-5) (CR-6 VAR-7)

RUN TIME 4 MIN 55.3 SEC

EXAM    TRY    FIRE   WMACT  E/F   E/T   T/F
2905    927    451    1807   6.44  3.13  2.06
0.102   0.319  0.655  0.164  SEC AVG

1040 INSERTS 767 DELETES 59 WARNINGS 56 NEW OBJECTS
MAX SMPX LENGTH 121
CORE (FREE.FULL): (5321 . 922) USED (5044 . 604)

FIRED 83 OUT OF 259 PRODS

TEST13
```

E.

```
 ((THE NUMBER OF SOLDIERS THE RUSSIANS HAVE IS ONE HALF OF THE NUMBER OF GUNS
      THEY HAVE . THE NUMBER OF GUNS THEY HAVE IS 7000 . HOW MANY SOLDIERS DO
      THEY HAVE ?))
ISEQN (C-1 (EQUAL VAR-1 (TIMES VAR-2 VAR-3))) (C-2 (EQUAL VAR-3 VAR-4))
HASREPR (VAR-1 (THE NUMBER OF SOLDIERS THE RUSSIANS HAVE)) (VAR-2 (0.5))
 (VAR-3 (THE NUMBER OF GUNS THEY HAVE)) (VAR-4 (7000))
EVLIST (PH-1 ((VAR-1)))
EQVARCHUNK (C-3 CL-1) (CL-3 CR-2)
HASEXPR (C-1 (EQUAL VAR-1 (TIMES VAR-2 VAR-3))) (C-2 (EQUAL VAR-3 VAR-4))
 (C-3 VAR-1) (CL-1 VAR-1) (CL-2 VAR-2) (CL-3 VAR-3) (CR-1 (TIMES VAR-2 VAR-3))
 (CR-2 VAR-3) (CR-3 VAR-4)

RUN TIME 3 MIN. 18.6 SEC

EXAM    TRY    FIRE   WMACT  E/F   E/T   T/F
4143    1209   441    1679   9.39  3.43  2.74
0.0479  0.164  0.450  0.118  SEC AVG

971 INSERTS 708 DELETES 37 WARNINGS 55 NEW OBJECTS
MAX :SMPX LENGTH 105
CORE (FREE,FULL): (4564 . 859) USED (5124 . 616)

FIRED 86 OUT OF 260 PRODS

TEST 14
 ((THE NUMBER OF STUDENTS WHO PASSED THE ADMISSIONS TEST IS 10 PER CENT OF THE
      TOTAL NUMBER OF STUDENTS IN THE HIGH SCHOOL . IF THE NUMBER OF
      SUCCESSFUL CANDIDATES IS 72 , WHAT IS THE NUMBER OF STUDENTS IN THE HIGH
      SCHOOL ?))
ISEQN (C-1 (EQUAL VAR-1 (TIMES VAR-2 VAR-3))) (C-2 (EQUAL VAR-4 VAR-5))
HASREPR (VAR-1 (THE NUMBER OF STUDENTS WHO PASSED THE ADMISSIONS TEST))
 (VAR-2 (0.09999999)) (VAR-3 (THE NUMBER OF STUDENTS IN THE HIGH SCHOOL))
 (VAR-4 (THE NUMBER OF SUCCESSFUL CANDIDATES)) (VAR-5 (72))
EVLIST (PH-1 ((VAR-3)))
EQVARCHUNK (C-3 CR-2)
HASEXPR (C-1 (EQUAL VAR-1 (TIMES VAR-2 VAR-3))) (C-2 (EQUAL VAR-4 VAR-5))
 (C-3 VAR-3) (CL-1 VAR-1) (CL-2 VAR-2) (CL-3 VAR-4) (CR-1 (TIMES VAR-2 VAR-3))
 (CR-2 VAR-3) (CR-3 VAR-5)

RUN TIME 3 MIN. 23.5 SEC

EXAM    TRY    FIRE   WMACT  E/F   E/T   T/F
4708    1146   444    1803   10.6  4.11  2.58
0.0432  0.178  0.458  0.113  SEC AVG

1045 INSERTS 758 DELETES 35 WARNINGS 63 NEW OBJECTS
MAX :SMPX LENGTH 104
CORE (FREE,FULL): (4785 . 840) USED (4903 . 635)

FIRED 81 OUT OF 259 PRODS

TEST 15
 ((THE DISTANCE FROM NEW YORK TO LOS ANGELES IS 3000 MILES . IF THE AVERAGE
      SPEED OF A JET PLANE IS 600 MILES PER HOUR , FIND THE TIME IT TAKES TO
      TRAVEL FROM NEW YORK TO LOS ANGELES BY JET .))
ISEQN (C-1 (EQUAL VAR-1 (TIMES VAR-2 VAR-3)))
 (C-2 (EQUAL VAR-4 (TIMES VAR-5 (QUOTIENT VAR-3 VAR-6))))
HASREPR (VAR-1 (THE DISTANCE FROM NEW YORK TO LOS ANGELES)) (VAR-2 (3000))
 (VAR-3 (MILE)) (VAR-4 (THE AVERAGE SPEED OF A JET PLANE)) (VAR-5 (600))
 (VAR-6 (HOUR))
 (VAR-7 (THE TIME IT TAKES TO TRAVEL FROM NEW YORK TO LOS ANGELES BY JET))
EVLIST (PH-1 ((VAR-7)))
EQVARCHUNK (CL-5 CR-2)
HASEXPR (C-1 (EQUAL VAR-1 (TIMES VAR-2 VAR-3)))
 (C-2 (EQUAL VAR-4 (TIMES VAR-5 (QUOTIENT VAR-3 VAR-6)))) (C-3 VAR-7)
 (CL-1 VAR-1) (CL-2 VAR-2) (CL-3 VAR-4) (CL-4 VAR-5) (CL-5 VAR-3)
 (CR-1 (TIMES VAR-2 VAR-3)) (CR-2 VAR-3)
 (CR-3 (TIMES VAR-5 (QUOTIENT VAR-3 VAR-6))) (CR-4 (QUOTIENT VAR-3 VAR-6))
 (CR-5 VAR-6)

RUN TIME 4 MIN. 14.8 SEC

EXAM    TRY    FIRE   WMACT  E/F   E/T   T/F
5011    1237   513    2091   9.77  4.05  2.41
0.0508  0.206  0.497  0.122  SEC AVG

1215 INSERTS 876 DELETES 45 WARNINGS 71 NEW OBJECTS
MAX :SMPX LENGTH 104
CORE (FREE,FULL): (4500 . 738) USED (5188 . 737)

FIRED 84 OUT OF 258 PRODS

TEST 16
 ((THE COST OF A BOX OF MIXED NUTS IS THE SUM OF THE COST OF THE ALMONDS IN THE
      BOX AND THE COST OF THE PECANS IN THE BOX . FOR A LARGE BOX THIS COST IS
      $ 3.5 . THE WEIGHT , IN POUNDS , OF A BOX OF MIXED NUTS IS THE SUM OF
      THE NUMBER OF POUNDS OF ALMONDS IN THE BOX AND THE NUMBER OF POUNDS OF
      PECANS IN THE BOX . THIS LARGE BOX WEIGHS 3 POUNDS . THE COST OF ALMONDS
      PER POUND OF ALMONDS IS $ 1 , AND THE COST OF PECANS PER POUND OF PECANS
      IS $ 1.5 . FIND THE COST OF THE ALMONDS IN THE BOX AND THE COST OF THE
      PECANS IN THE BOX .))
ISEQN (C-1 (EQUAL VAR-1 (PLUSS VAR-2 VAR-3)))
 (C-2 (EQUAL VAR-1 (TIMES VAR-4 VAR-5)))
 (C-3 (EQUAL VAR-6 (PLUSS VAR-7 VAR-8)))
 (C-5 (EQUAL (QUOTIENT VAR-2 VAR-7) (TIMES VAR-10 VAR-5)))
 (C-6 (EQUAL (QUOTIENT VAR-3 VAR-8) (TIMES VAR-11 VAR-5)))
 (CN-1 (EQUAL VAR-6 VAR-9))
HASREPR (VAR-1 (THE COST OF A BOX OF MIXED NUTS)) (VAR-10 (1)) (VAR-11 (1.5))
 (VAR-2 (THE COST OF THE ALMONDS IN THE BOX))
 (VAR-3 (THE COST OF THE PECANS IN THE BOX)) (VAR-4 (3.5)) (VAR-5 (DOLLAR))
 (VAR-6 (THE WEIGHT IN POUNDS OF A BOX OF MIXED NUTS))
 (VAR-7 (THE NUMBER OF POUNDS OF ALMONDS IN THE BOX))
 (VAR-8 (THE NUMBER OF POUNDS OF PECANS IN THE BOX)) (VAR-9 (3))
EVLIST (PH-1 ((VAR-2 VAR-3)))
EQVARCHUNK (C-7 CL-2) (C-8 CR-2) (CL-12 CR-2) (CL-9 CL-7) (CR-10 CR-4)
 (CR-12 CR-6) (CR-13 CR-8) (CR-9 CL-6)
HASEXPR (C-1 (EQUAL VAR-1 (PLUSS VAR-2 VAR-3)))
 (C-2 (EQUAL VAR-1 (TIMES VAR-4 VAR-5)))
 (C-3 (EQUAL VAR-6 (PLUSS VAR-7 VAR-8)))
 (C-5 (EQUAL (QUOTIENT VAR-2 VAR-7) (TIMES VAR-10 VAR-5)))
 (C-6 (EQUAL (QUOTIENT VAR-3 VAR-8) (TIMES VAR-11 VAR-5))) (C-7 VAR-2)
 (C-8 VAR-3) (CL-1 VAR-1) (CL-10 VAR-10) (CL-11 (QUOTIENT VAR-3 VAR-8))
 (CL-12 VAR-3) (CL-13 VAR-11) (CL-2 VAR-2) (CL-3 VAR-1) (CL-4 VAR-4)
 (CL-5 VAR-6) (CL-6 VAR-7) (CL-7 VAR-6) (CL-8 (QUOTIENT VAR-2 VAR-7))
 (CL-9 VAR-2) (CN-1 (EQUAL VAR-6 VAR-9)) (CR-1 (PLUSS VAR-2 VAR-3))
 (CR-10 VAR-5) (CR-11 (TIMES VAR-11 VAR-5)) (CR-12 VAR-8) (CR-13 VAR-5)
 (CR-2 VAR-3) (CR-3 (TIMES VAR-4 VAR-5)) (CR-4 VAR-5)
 (CR-5 (PLUSS VAR-7 VAR-8)) (CR-6 VAR-8) (CR-7 VAR-9)
 (CR-8 (TIMES VAR-10 VAR-5)) (CR-9 VAR-7)

RUN TIME 20 MIN. 6.20 SEC

EXAM    TRY    FIRE   WMACT  E/F   E/T   T/F
13991   4749   1948   6773   7.20  2.95  2.44
0.0862  0.254  0.620  0.178  SEC AVG

3769 INSERTS 3004 DELETES 280 WARNINGS 194 NEW OBJECTS
MAX :SMPX LENGTH 135
CORE (FREE,FULL): (7767 . 1082) USED (319 . -75)

FIRED 100 OUT OF 258 PRODS

TEST 17
 ((THE SUM OF TWO NUMBERS IS 96 , AND ONE NUMBER IS 16 LARGER THAN THE OTHER
      NUMBER . FIND THE TWO NUMBERS .))
ISEQN (C-1 (EQUAL (PLUSS VAR-1 VAR-2) VAR-3))
 (C-2 (EQUAL VAR-1 (PLUS VAR-4 VAR-2)))
HASREPR (VAR-1 (FIRST NUMBER)) (VAR-2 (THE SECOND NUMBER)) (VAR-3 (96))
 (VAR-4 (16))
EVLIST (PH-1 ((VAR-1 VAR-2)))
EQVARCHUNK (C-3 CL-2) (C-4 CR-2) (CL-3 CL-2) (CR-4 CR-2)
HASEXPR (C-1 (EQUAL (PLUSS VAR-1 VAR-2) VAR-3))
 (C-2 (EQUAL VAR-1 (PLUS VAR-4 VAR-2))) (C-3 VAR-1) (C-4 VAR-2)
 (CL-1 (PLUSS VAR-1 VAR-2)) (CL-2 VAR-1) (CL-3 VAR-1) (CL-4 VAR-4) (CR-1 VAR-3)
 (CR-2 VAR-2) (CR-3 (PLUS VAR-4 VAR-2)) (CR-4 VAR-2)

RUN TIME 2 MIN. 54.9 SEC

EXAM    TRY    FIRE   WMACT  E/F   E/T   T/F
3521    877    394    1614   8.94  4.01  2.23
0.0497  0.199  0.444  0.108  SEC AVG

939 INSERTS 675 DELETES 46 WARNINGS 55 NEW OBJECTS
MAX :SMPX LENGTH 138
CORE (FREE,FULL): (6104 . 898) USED (2178 . 518)

FIRED 88 OUT OF 258 PRODS

TEST 18
 ((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 ?))
ISEQN (C-1 (EQUAL VAR-1 (TIMES VAR-2 (QUOTIENT VAR-3 VAR-4))))
 (C-2 (EQUAL VAR-5 (TIMES VAR-6 VAR-3)))
HASREPR (VAR-1 (THE GAS CONSUMPTION OF MY CAR)) (VAR-2 (15)) (VAR-3 (MILE))
 (VAR-4 (GALLON)) (VAR-5 (THE DISTANCE BETWEEN BOSTON AND NEW YORK))
```

```
(VAR.6 (250))
(VAR.7
 (THE NUMBER OF GALLONS OF GAS USED ON A TRIP BETWEEN NEW YORK AND BOSTON))
EVLIST (PB.1 ((VAR.7)))
EQVARCHUNK ((CR.5 CL.3))
HASEXPR (C.1 (EQUAL VAR.1 (TIMES VAR.2 (QUOTIENT VAR.3 VAR.4))))
 (C.2 (EQUAL VAR.5 (TIMES VAR.6 VAR.3))) (C.3 VAR.7) (CL.1 VAR.1) (CL.2 VAR.2)
 (CL.3 VAR.3) (CL.4 VAR.5) (CL.5 VAR.6)
 (CR.1 (TIMES VAR.2 (QUOTIENT VAR.3 VAR.4))) (CR.2 (QUOTIENT VAR.3 VAR.4))
 (CR.3 VAR.4) (CR.4 (TIMES VAR.6 VAR.3)) (CR.5 VAR.3)

RUN TIME 4 MIN 58.2 SEC

EXAM    TRY    FIRE   WMACT  E/T   E/F   T/F
5003    1211   512    2952   9.77  4.13  2.37
0.0596  0.246  0.582  0.165  SEC AVG

1193 INSERTS 858 DELETES 48 WARNINGS 70 NEW OBJECTS
MAX :SMEX LENGTH 147
CORE (FREE.FULL): (4529 . 690) USED (3705 . 727).

FIRED 83 OUT OF 258 PRODS

TEST19
((THE DAILY COST OF LIVING FOR A GROUP IS THE OVERHEAD COST PLUS THE RUNNING
    COST FOR EACH PERSON TIMES THE NUMBER OF PEOPLE IN THE GROUP . THIS COST
    FOR ONE GROUP EQUALS $ 100 . AND THE NUMBER OF PEOPLE IN THE GROUP IS 40
    . IF THE OVERHEAD COST IS 10 TIMES THE RUNNING COST . FIND THE OVERHEAD
    AND THE RUNNING COST FOR EACH PERSON .))
ISEQN (C.1 (EQUAL VAR.1 (PLUS VAR.2 (TIMES VAR.3 VAR.4))))
 (C.2 (EQUAL VAR.1 (TIMES VAR.5 VAR.6))) (C.3 (EQUAL VAR.4 VAR.7))
 (C.4 (EQUAL VAR.2 (TIMES VAR.8 VAR.3)))
HASREFR (VAR.1 (THE DAILY COST OF LIVING FOR A GROUP))
 (VAR.2 (THE OVERHEAD COST)) (VAR.3 (THE RUNNING COST FOR EACH PERSON))
 (VAR.4 (THE NUMBER OF PEOPLE IN THE GROUP)) (VAR.5 (100)) (VAR.6 (DOLLAR))
 (VAR.7 (40)) (VAR.8 (10))
EVLIST (PB.1 ((VAR.2 VAR.3)))
EQVARCHUNK (C.5 CL.2) (C.6 CL.3) (CL.6 CR.3) (CL.7 CL.2) (CR.8 CL.3)
HASEXPR (C.1 (EQUAL VAR.1 (PLUS VAR.2 (TIMES VAR.3 VAR.4))))
 (C.2 (EQUAL VAR.1 (TIMES VAR.5 VAR.6))) (C.3 (EQUAL VAR.4 VAR.7))
 (C.4 (EQUAL VAR.2 (TIMES VAR.8 VAR.3))) (C.5 VAR.2) (C.6 VAR.3) (CL.1 VAR.1)
 (CL.2 VAR.2) (CL.3 VAR.3) (CL.4 VAR.1) (CL.5 VAR.5) (CL.6 VAR.4) (CL.7 VAR.2)
 (CL.8 VAR.8) (CR.1 (PLUS VAR.2 (TIMES VAR.3 VAR.4)))
 (CR.2 (TIMES VAR.3 VAR.4)) (CR.3 VAR.4) (CR.4 (TIMES VAR.5 VAR.6))
 (CR.5 VAR.6) (CR.6 VAR.7) (CR.7 (TIMES VAR.8 VAR.3)) (CR.8 VAR.3)

RUN TIME 7 MIN 45.8 SEC

EXAM    TRY    FIRE   WMACT  E/T   E/F   T/F
8397    2566   1018   3709   8.25  3.27  2.52
0.0565  0.182  0.458  0.126  SEC AVG

2109 INSERTS 1629 DELETES 118 WARNINGS 119 NEW OBJECTS
MAX :SMEX LENGTH 107
CORE (FREE.FULL): (8022 . 1605) USED (11370 . 1612)

FIRED 93 OUT OF 260 PRODS

TEST20
((THE RUSSIAN ARMY HAS 6 TIMES AS MANY RESERVES IN A UNIT AS IT HAS UNIFORMED
    SOLDIERS . THE PAY FOR RESERVES EACH MONTH IS 50 DOLLARS TIMES THE
    NUMBER OF RESERVES IN THE UNIT . AND THE AMOUNT SPENT ON THE REGULAR
    ARMY EACH MONTH IS $ 150 TIMES THE NUMBER OF UNIFORMED SOLDIERS . THE
    SUM OF THIS LATTER AMOUNT AND THE PAY FOR RESERVES EACH MONTH EQUALS $
    45000 . FIND THE NUMBER OF RESERVES IN A UNIT THE RUSSIAN ARMY HAS AND
    THE NUMBER OF UNIFORMED SOLDIERS IT HAS .))
ISEQN (C.2 (EQUAL VAR.4 (TIMES VAR.5 (TIMES VAR.6 VAR.2))))
 (C.3 (EQUAL VAR.7 (TIMES VAR.8 (TIMES VAR.6 VAR.3))))
 (C.4 (EQUAL (PLUSS VAR.7 VAR.4) (TIMES VAR.9 VAR.6)))
 (CN.1 (EQUAL (TIMES VAR.1 VAR.2) VAR.3)))
HASREFR (VAR.1 (6))
 (VAR.2 (THE NUMBER OF RESERVES IN A UNIT THE RUSSIAN ARMY HAS))
 (VAR.3 (THE NUMBER OF UNIFORMED SOLDIERS IT HAS))
 (VAR.4 (THE PAY FOR RESERVES EACH MONTH)) (VAR.5 (50)) (VAR.6 (DOLLAR))
 (VAR.7 (THE AMOUNT SPENT ON THE REGULAR ARMY EACH MONTH)) (VAR.8 (150))
 (VAR.9 (45000))
EVLIST (PB.1 ((VAR.2 VAR.3)))
EQVARCHUNK (C.5 CR.2) (C.6 CR.1) (CL.8 CL.5) (CR.10 CL.3) (CR.11 CL.5)
 (CR.5 CR.2) (CR.8 CR.1)
HASEXPR (C.2 (EQUAL VAR.4 (TIMES VAR.5 (TIMES VAR.6 VAR.2))))
 (C.3 (EQUAL VAR.7 (TIMES VAR.8 (TIMES VAR.6 VAR.3))))
 (C.4 (EQUAL (PLUSS VAR.7 VAR.4) (TIMES VAR.9 VAR.6))) (C.5 VAR.2) (C.6 VAR.3)
 (CL.1 (TIMES VAR.1 VAR.2)) (CL.10 VAR.7) (CL.11 VAR.9) (CL.2 VAR.1)
```

```
 (CL.3 VAR.4) (CL.4 VAR.5) (CL.5 VAR.6) (CL.6 VAR.7) (CL.7 VAR.8) (CL.8 VAR.6)
 (CL.9 (PLUSS VAR.7 VAR.4)) (CN.1 (EQUAL (TIMES VAR.1 VAR.2) VAR.3))
 (CR.1 VAR.3) (CR.10 VAR.4) (CR.11 VAR.6) (CR.2 VAR.2)
 (CR.3 (TIMES VAR.5 (TIMES VAR.6 VAR.2))) (CR.4 (TIMES VAR.6 VAR.2))
 (CR.5 VAR.2) (CR.6 (TIMES VAR.8 (TIMES VAR.6 VAR.3)))
 (CR.7 (TIMES VAR.6 VAR.3)) (CR.8 VAR.3) (CR.9 (TIMES VAR.9 VAR.6))

RUN TIME 14 MIN 21.5 SEC

EXAM    TRY    FIRE   WMACT  E/T   E/F   T/F
11279   3691   1420   5127   7.94  3.06  2.60
0.0764  0.233  0.607  0.168  SEC AVG

2893 INSERTS 2234 DELETES 178 WARNINGS 152 NEW OBJECTS
MAX :SMEX LENGTH 148
CORE (FREE.FULL): (5193 . 902) USED (14199 . 2315)

FIRED 99 OUT OF 258 PRODS

TEST21
((THE SUM OF TWO NUMBERS IS TWICE THE DIFFERENCE BETWEEN THE TWO NUMBERS . THE
    FIRST NUMBER EXCEEDS THE SECOND NUMBER BY 5 . FIND THE TWO NUMBERS .))
ISEQN (C.1 (EQUAL (PLUSS VAR.1 VAR.2) (TIMES VAR.3 (MINUSS VAR.1 VAR.2))))
 (C.2 (EQUAL (MINUS VAR.1 VAR.2) VAR.4)))
HASREFR (VAR.1 (FIRST NUMBER)) (VAR.2 (THE SECOND NUMBER)) (VAR.3 (2))
 (VAR.4 (5))
EVLIST (PB.1 ((VAR.1 VAR.2)))
EQVARCHUNK (C.3 CL.2) (C.4 CR.2) (CL.4 CL.2) (CL.6 CL.2) (CR.4 CR.2) (CR.6 CR.2)
HASEXPR (C.1 (EQUAL (PLUSS VAR.1 VAR.2) (TIMES VAR.3 (MINUSS VAR.1 VAR.2))))
 (C.2 (EQUAL (MINUS VAR.1 VAR.2) VAR.4)) (C.3 VAR.1) (C.4 VAR.2)
 (CL.1 (PLUSS VAR.1 VAR.2)) (CL.2 VAR.1) (CL.3 VAR.3) (CL.4 VAR.1)
 (CL.5 (MINUS VAR.1 VAR.2)) (CL.6 VAR.1)
 (CR.1 (TIMES VAR.3 (MINUSS VAR.1 VAR.2))) (CR.2 VAR.2)
 (CR.3 (MINUSS VAR.1 VAR.2)) (CR.4 VAR.2) (CR.5 VAR.4) (CR.6 VAR.2)

RUN TIME 3 MIN 18.4 SEC

EXAM    TRY    FIRE   WMACT  E/T   E/F   T/F
4740    1331   569    2258   8.33  3.56  2.34
0.0418  0.149  0.349  0.0879 SEC AVG

1278 INSERTS 962 DELETES 29 WARNINGS 20 NEW OBJECTS
MAX :SMEX LENGTH 141
CORE (FREE.FULL): (12797 . 2416) USED (6595 . 801)

FIRED 83 OUT OF 260 PRODS

TEST22
((THE SUM OF TWO NUMBERS IS 111 . ONE OF THE NUMBERS IS CONSECUTIVE TO THE
    OTHER NUMBER . FIND THE TWO NUMBERS .))
ISEQN (C.1 (EQUAL (PLUSS VAR.1 VAR.2) VAR.3))
 (C.2 (EQUAL VAR.1 (PLUS VAR.4 VAR.2)))
HASREFR (VAR.1 (FIRST NUMBER)) (VAR.2 (THE SECOND NUMBER)) (VAR.3 (111))
 (VAR.4 (1))
EVLIST (PB.1 ((VAR.1 VAR.2)))
EQVARCHUNK (C.3 CL.2) (C.4 CR.2) (CL.3 CL.2) (CR.4 CR.2)
HASEXPR (C.1 (EQUAL (PLUSS VAR.1 VAR.2) VAR.3))
 (C.2 (EQUAL VAR.1 (PLUS VAR.4 VAR.2))) (C.3 VAR.1) (C.4 VAR.2)
 (CL.1 (PLUSS VAR.1 VAR.2)) (CL.2 VAR.1) (CL.3 VAR.1) (CL.4 VAR.4) (CR.1 VAR.3)
 (CR.2 VAR.2) (CR.3 (PLUS VAR.4 VAR.2)) (CR.4 VAR.2)

RUN TIME 3 MIN 3.42 SEC

EXAM    TRY    FIRE   WMACT  E/T   E/F   T/F
3686    948    402    1620   9.17  3.89  2.36
0.0498  0.193  0.456  0.113  SEC AVG

944 INSERTS 676 DELETES 47 WARNINGS 56 NEW OBJECTS
MAX :SMEX LENGTH 139
CORE (FREE.FULL): (5763 . 829) USED (3448 . 536)

FIRED 89 OUT OF 258 PRODS

TEST23
((THE SUM OF THREE NUMBERS IS 9 . THE SECOND NUMBER IS 3 MORE THAN 2 TIMES THE
    FIRST NUMBER . THE THIRD NUMBER EQUALS THE SUM OF THE FIRST TWO NUMBERS
    . FIND THE THREE NUMBERS .))
ISEQN (C.1 (EQUAL (PLUSS VAR.1 (PLUSS VAR.2 VAR.3)) VAR.4))
 (C.2 (EQUAL VAR.2 (PLUS VAR.5 (TIMES VAR.6 VAR.1))))
 (C.3 (EQUAL VAR.3 (PLUSS VAR.1 VAR.2)))
HASREFR (VAR.1 (THE FIRST NUMBER)) (VAR.2 (THE SECOND NUMBER))
 (VAR.3 (THE THIRD NUMBER)) (VAR.4 (9)) (VAR.5 (3)) (VAR.6 (2))
EVLIST (PB.1 ((VAR.1 VAR.2 VAR.3)))
```

```
EQVARCHUNK ((C-4 CL-2) (C-5 CL-3) (C-6 CR-3) (CL-4 CL-3) (CL-7 CR-3) (CL-8 CL-2)
 (CR-6 CL-2) (CR-8 CL-3))
HASEXPR (C-1 (EQUAL (PLUSS VAR-1 (PLUSS VAR-2 VAR-3)) VAR-4))
 (C-2 (EQUAL VAR-2 (PLUS VAR-5 (TIMES VAR-6 VAR-1))))
 (C-3 (EQUAL VAR-3 (PLUSS VAR-1 VAR-2))) (C-4 VAR-1) (C-5 VAR-2) (C-6 VAR-3)
 (CL-1 (PLUSS VAR-1 (PLUSS VAR-2 VAR-3))) (CL-2 VAR-1) (CL-3 VAR-2)
 (CL-4 VAR-2) (CL-5 VAR-5) (CL-6 VAR-6) (CL-7 VAR-3) (CL-8 VAR-1) (CR-1 VAR-4)
 (CR-2 (PLUSS VAR-2 VAR-3)) (CR-3 VAR-3)
 (CR-4 (PLUS VAR-5 (TIMES VAR-6 VAR-1))) (CR-5 (TIMES VAR-6 VAR-1))
 (CR-6 VAR-1) (CR-7 (PLUSS VAR-1 VAR-2)) (CR-8 VAR-2)

RUN TIME 6 MIN 38.2 SEC

EXAM    TRY    FIRE   WMACT  E/F   E/T   T/F
6664    1957   883    3706   7.55  3.41  2.22
0.0506  0.185  0.012  0.112  SEC AVG

1840 INSERTS 1002 DELETES 134 WARNINGS 97 NEW OBJECTS
MAX SMPX LENGTH 109
CORE (FREE.FULL): (8552 . 736) USED (9430 . 1364)

FIRED 93 OUT OF 259 PRODS

TEST24
 ((THE SUM OF THREE NUMBERS IS 100 . THE THIRD NUMBER EQUALS THE SUM OF THE
     FIRST TWO NUMBERS . THE DIFFERENCE BETWEEN THE FIRST TWO NUMBERS IS 10
     PERCENT OF THE THIRD NUMBER . FIND THE THREE NUMBERS .))
ISEQN (C-1 (EQUAL (PLUSS VAR-1 (PLUSS VAR-2 VAR-3)) VAR-4))
 (C-2 (EQUAL VAR-3 (PLUSS VAR-1 VAR-2)))
 (C-3 (EQUAL (MINUSS VAR-1 VAR-2) (TIMES VAR-5 VAR-3)))
HASREFR (VAR-1 (THE FIRST NUMBER)) (VAR-2 (THE SECOND NUMBER))
 (VAR-3 (THE THIRD NUMBER)) (VAR-4 (100)) (VAR-5 (0.09999999)))
EVLIST (PH-1 ((VAR-1 VAR-2 VAR-3)))
EQVARCHUNK ((C-4 CL-2) (C-5 CL-3) (C-6 CR-3) (CL-4 CR-3) (CL-5 CL-2) (CL-7 CL-2)
 (CR-5 CL-3) (CR-7 CL-3) (CR-8 CR-3))
HASEXPR (C-1 (EQUAL (PLUSS VAR-1 (PLUSS VAR-2 VAR-3)) VAR-4))
 (C-2 (EQUAL VAR-3 (PLUSS VAR-1 VAR-2)))
 (C-3 (EQUAL (MINUSS VAR-1 VAR-2) (TIMES VAR-5 VAR-3))) (C-4 VAR-1) (C-5 VAR-2)
 (C-6 VAR-3) (CL-1 (PLUSS VAR-1 (PLUSS VAR-2 VAR-3))) (CL-2 VAR-1) (CL-3 VAR-2)
 (CL-4 VAR-3) (CL-5 VAR-1) (CL-6 (MINUSS VAR-1 VAR-2)) (CL-7 VAR-1)
 (CL-8 VAR-5) (CR-1 VAR-4) (CR-2 (PLUSS VAR-2 VAR-3)) (CR-3 VAR-3)
 (CR-4 (PLUSS VAR-1 VAR-2)) (CR-5 VAR-2) (CR-6 (TIMES VAR-5 VAR-3))
 (CR-7 VAR-2) (CR-8 VAR-3)

RUN TIME 6 MIN 57.8 SEC

EXAM    TRY    FIRE   WMACT  E/F   E/T   T/F
6835    1997   886    3168   7.71  3.42  2.25
0.0535  0.183  0.012  0.109  SEC AVG

1906 INSERTS 1032 DELETES 108 WARNINGS 104 NEW OBJECTS
MAX SMPX LENGTH 119
CORE (FREE.FULL): (9028 . 731) USED (1564 . 859)

FIRED 93 OUT OF 258 PRODS

TEST25
 ((  C EQUALS B TIMES D PLUS 1 . AND B PLUS D EQUALS 3 . AND B MINUS D EQUALS
     1 . FIND C .))
ISEQN (C-1 (EQUAL VAR-1 (PLUS (TIMES VAR-2 VAR-3) VAR-4)))
 (C-2 (EQUAL (PLUS VAR-2 VAR-3) VAR-5)) (C-3 (EQUAL (MINUS VAR-2 VAR-3) VAR-4))
HASREFR (VAR-1 (C)) (VAR-2 (B)) (VAR-3 (D)) (VAR-4 (1)) (VAR-5 (3))
EVLIST (PH-1 ((VAR-1)))
EQVARCHUNK (C-4 CL-1) (CL-5 CL-3) (CL-7 CL-3) (CR-5 CR-3) (CR-6 CR-2)
 (CR-7 CR-3)
HASEXPR (C-1 (EQUAL VAR-1 (PLUS (TIMES VAR-2 VAR-3) VAR-4)))
 (C-2 (EQUAL (PLUS VAR-2 VAR-3) VAR-5)) (C-3 (EQUAL (MINUS VAR-2 VAR-3) VAR-4))
 (C-4 VAR-1) (CL-1 VAR-1) (CL-2 (TIMES VAR-2 VAR-3)) (CL-3 VAR-2)
 (CL-4 (PLUS VAR-2 VAR-3)) (CL-5 VAR-2) (CL-6 (MINUS VAR-2 VAR-3)) (CL-7 VAR-2)
 (CR-1 (PLUS (TIMES VAR-2 VAR-3) VAR-4)) (CR-2 VAR-4) (CR-3 VAR-3) (CR-4 VAR-5)
 (CR-5 VAR-3) (CR-6 VAR-4) (CR-7 VAR-3)

RUN TIME 4 MIN 27.6 SEC

EXAM    TRY    FIRE   WMACT  E/F   E/T   T/F
3287    1009   519    1969   6.33  3.26  1.94
0.0818  0.265  0.516  0.136  SEC AVG

1135 INSERTS 836 DELETES 84 WARNINGS 55 NEW OBJECTS
MAX SMPX LENGTH 131
CORE (FREE.FULL): (4430 . 879) USED (6052 . 719)

FIRED 84 OUT OF 260 PRODS
```

```
TEST26
 ((THE SQUARE OF THE DIFFERENCE BETWEEN THE NUMBER OF APPLES AND THE NUMBER OF
     ORANGES ON THE TABLE IS EQUAL TO 9 . IF THE NUMBER OF APPLES IS 7 . FIND
     THE NUMBER OF ORANGES ON THE TABLE .))
ISEQN (C-1 (EQUAL (EXPT (MINUSS VAR-1 VAR-2) VAR-3) VAR-4))
 (C-2 (EQUAL VAR-1 VAR-5))
HASREFR (VAR-1 (THE NUMBER OF APPLES))
 (VAR-2 (THE NUMBER OF ORANGES ON THE TABLE)) (VAR-3 (2)) (VAR-4 (9))
 (VAR-5 (7))
EVLIST (PH-1 ((VAR-2)))
EQVARCHUNK ((C-3 CR-3) (CL-4 CL-3)
HASEXPR (C-1 (EQUAL (EXPT (MINUSS VAR-1 VAR-2) VAR-3) VAR-4))
 (C-2 (EQUAL VAR-1 VAR-5)) (C-3 VAR-2) (CL-1 (EXPT (MINUSS VAR-1 VAR-2) VAR-3))
 (CL-2 (MINUSS VAR-1 VAR-2)) (CL-3 VAR-1) (CL-4 VAR-1) (CR-1 VAR-4)
 (CR-2 VAR-3) (CR-3 VAR-2) (CR-4 VAR-5)

RUN TIME 3 MIN. 36.4 SEC

EXAM    TRY    FIRE   WMACT  E/F   E/T   T/F
4070    1136   459    1856   8.88  3.59  2.47
0.0531  0.190  0.071  0.117  SEC AVG

1070 INSERTS 786 DELETES 44 WARNINGS 63 NEW OBJECTS
MAX SMPX LENGTH 153
CORE (FREE.FULL): (4895 . 913) USED (5587 . 635)

FIRED 88 OUT OF 260 PRODS

TEST27
 ((THE GROSS WEIGHT OF A SHIP IS 20000 TONS . IF ITS NET WEIGHT IS 15000 TONS .
     WHAT IS THE WEIGHT OF THE SHIPS CARGO ?))
ISEQN (C-1 (EQUAL VAR-1 (TIMES VAR-2 VAR-3)))
 (C-2 (EQUAL VAR-4 (TIMES VAR-5 VAR-3)))
HASREFR (VAR-1 (THE GROSS WEIGHT OF A SHIP)) (VAR-2 (20000)) (VAR-3 (TON))
 (VAR-4 (ITS NET WEIGHT)) (VAR-5 (15000))
 (VAR-6 (THE WEIGHT OF THE SHIPS CARGO))
EVLIST (PH-1 ((VAR-6)))
EQVARCHUNK ((CR-4 CR-2)
HASEXPR (C-1 (EQUAL VAR-1 (TIMES VAR-2 VAR-3)))
 (C-2 (EQUAL VAR-4 (TIMES VAR-5 VAR-3))) (C-3 VAR-6) (CL-1 VAR-1) (CL-2 VAR-2)
 (CL-3 VAR-4) (CL-4 VAR-5) (CR-1 (TIMES VAR-2 VAR-3)) (CR-2 VAR-3)
 (CR-3 (TIMES VAR-5 VAR-3)) (CR-4 VAR-3)

RUN TIME 2 MIN 52.3 SEC

EXAM    TRY    FIRE   WMACT  E/F   E/T   T/F
3454    852    381    1550   9.07  4.05  2.24
0.0499  0.207  0.052  0.111  SEC AVG

905 INSERTS 605 DELETES 38 WARNINGS 53 NEW OBJECTS
MAX SMPX LENGTH 139
CORE (FREE.FULL): (5574 . 1037) USED (6908 . 511)

FIRED 79 OUT OF 260 PRODS
```

## Appendix F. LISTING OF THE KNOWLEDGE STATEMENTS

```
KNOWLEDGE STATEMENTS AND USES. ORDER:
   NS NI ND NP NA NC NP NQ NE NW NO NT 9 21.

[illegible] NS [illegible]

 NS1   THE FIRST FUNCTION OF THE INITIAL SCAN IS TO APPLY TRANSFORMATIONS AT
          EACH POINT IN THE SCAN.
       USED-FOR ALL.
       IS-FOR    S10 S13 S15 S40 S65 I1 I2 I2E I3 I4 I5 I6 I7 I8 I9 I9E I10
                    I11 I12 I13 I14 I15 I16 I17 I18 I19 I20 I21 I22 I23 I24
                    I25 I26 I26D I27 I28 I29 I30 I31 I32 I50 I51 A10.
 NS2   THE SECOND FUNCTION OF THE INITIAL SCAN IS TO APPLY AGE PROBLEM
          TRANSFORMATIONS, IF THE PROBLEM IS AN AGE PROBLEM, AT EACH SCAN
          POINT.
       USED-FOR ALL AGE TEST11 TEST12 TEST13 TEST113 TEST110.
       IS-FOR    S16 I50 I51 A11 A12 A15 A17 A18 A19 A20 A24 A26 A28 A31 A32
                    A34 A36 A38 A41 A50.
 NS3   THE THIRD FUNCTION OF THE INITIAL SCAN IS TO PUT DICTIONARY TAGS ON
          WORDS AS EACH WORD IS SCANNED.
       USED-FOR ALL.
       IS-FOR    S12 S10 D1 D3 D5 D7 D9 D11 D13 D14 D15 D17 D18 D19 D21 D24
                    D27 D30 D41 D44 D47 D50 D53 D55 D57 D61 D63 D65 D67 D675
                    D69 D71 D715 D72 D73 D75 D77 D78 D79 D81 D83 D85 D87 D91
                    D95.
 NS4   THE FOURTH FUNCTION OF THE INITIAL SCAN IS TO CHECK FOR A NEW HIGH
          PRECEDENCE WITHIN THE CHUNK BEING SCANNED, IF THAT CHUNK IS NOT AN
          EV CHUNK, AS EACH WORD IS SCANNED.
       USED-FOR ALL.
       IS-FOR    S20 S25 S30 S40.
 NS5   THE FOURTH FUNCTION OF THE INITIAL SCAN IS TO APPLY THE EV
          TRANSFORMATIONS, IF THE CHUNK BEING SCANNED IS AN EV CHUNK, AS EACH
          WORD IS SCANNED; AN EV TRANSFORMATION IS ANY OPERATION THAT DEALS
          WITH THE DETERMINATION OF EV CHUNKS.
       USED-FOR ALL.
       IS-FOR    S15 E5 E15 E20 E25 E35 E40 E45 E48 E50 E52 E55 E60 E70 E75.
 NS6   A CHUNK THAT STARTS WITH A WORD THAT IS A QWORD IS AN EV CHUNK.
       USED-FOR ALL.
       IS-FOR    S60.
 NS7   WHEN A PERIOD WITH A DELIMITER TAG IS SCANNED, THE END OF THE CURRENT
          CHUNK HAS BEEN REACHED, IF THE CHUNK IS NOT AN EV CHUNK.
       USED-FOR ALL.
       IS-FOR    S40.
       INTERP    S10 S13.
 NS8   THE FIRST CHUNK TO BE SCANNED STARTS IMMEDIATELY TO THE RIGHT OF THE
          LEFT END OF THE PROBLEM STRING.
       USED-FOR ALL.
       IS-FOR    S10.
 NS9   WHEN THE END OF ONE CHUNK IS REACHED, ANOTHER BEGINS IMMEDIATELY,
          UNLESS THE RIGHT END OF THE PROBLEM STRING HAS BEEN REACHED.
       USED-FOR ALL.
       IS-FOR    S40 S65.
 NS10  THE PERIOD AT THE END OF A CHUNK IS NOT INCLUDED AS PART OF THAT CHUNK
          OR ANY OTHER CHUNK.
       USED-FOR ALL.
       IS-FOR    S40.
       INTERP    S10 S13 E20 E30 E40 E70.
 NS11  THE INITIAL SCAN PROCEEDS FROM LEFT TO RIGHT IN THE PROBLEM STRING,
          PERFORMING THE FOUR FUNCTIONS AT EACH POINT IN TURN, AND ADDING
          EACH WORD SCANNED TO THE CURRENT CHUNK.
       USED-FOR ALL.
       IS-FOR    S10 S13 S15 S40.
       INTERP    E20 E35 E40 E50.
 NS12  WHEN THE END OF A CHUNK IS SCANNED, THE CHUNK IS COMPLETE, AND THE
          INITIAL SCAN IS INTERRUPTED FOR THE CHUNK SPLITTING PROCESS.
       USED-FOR ALL.
       IS-FOR    S40 E15 E20 E35 E70 E75.
       INTERP    S10 S13.
 NS13  THE LAST CHUNK IN A PROBLEM IS ALWAYS AN EV CHUNK.
       USED-FOR ALL.
       IS-FOR    S65 S70.
       INTERP    S40.
 NS14  WHEN THE END OF THE PROBLEM STRING IS REACHED, THE ANSWER BUILDING
          PROCESS MUST BE INITIATED.
       USED-FOR ALL.
       IS-FOR    S70.

[illegible] NI [illegible]

 NI1   "HOW OLD" TRANSFORMS TO "WHAT".
       USED-FOR TEST6 TEST9.
       IS-FOR    I1.
       INTERP    D87.
 NI2   "IS EQUAL TO" AND "EQUALS" TRANSFORM TO "IS".
       USED-FOR TEST2 TEST17 TEST19 TEST20 TEST23 TEST24 TEST25 TEST26.
       IS-FOR    I2 I2E.
 NI3   "YEARS YOUNGER THAN" TRANSFORMS TO "LESSTHAN".
       USED-FOR.
       IS-FOR    I3.
 NI4   "YEARS OLDER THAN" TRANSFORMS TO "PLUS".
       USED-FOR TEST9.
       IS-FOR    I4.
 NI5   "PER CENT LESS THAN" , PRECEDED BY A NUMBER, TRANSFORMS TO "TIMES" ,
          AND THE NUMBER IS REPLACED BY (100 - NUMBER) / 100.
       USED-FOR TEST8.
       IS-FOR    I5.
       INTERP    I16.
 NI6   "LESS THAN" TRANSFORMS TO "LESSTHAN".
       USED-FOR.
       IS-FOR    I6.
 NI7   "THESE" TRANSFORMS TO "THE".
       USED-FOR.
       IS-FOR    I7.
 NI8   "MORE THAN" TRANSFORMS TO "PLUS".
       USED-FOR TEST23.
       IS-FOR    I8.
 NI9   "FIRST TWO NUMBERS" AND "TWO NUMBERS" TRANSFORM TO "FIRST NUMBER AND
          THE SECOND NUMBER".
       USED-FOR TEST17 TEST21 TEST22 TEST23 TEST24.
       IS-FOR    I9 I9E.
 NI10  "THREE NUMBERS" TRANSFORMS TO "FIRST NUMBER AND THE SECOND NUMBER AND
          THE THIRD NUMBER".
       USED-FOR TEST23 TEST24.
       IS-FOR    I10.
 NI11  "ONE HALF" TRANSFORMS TO "0.5 ".
       USED-FOR TEST2 TEST9 TEST13.
       IS-FOR    I11.
 NI12  "TWICE" TRANSFORMS TO "2 TIMES".
       USED-FOR TEST2 TEST4 TEST5 TEST6 TEST7 TEST10 TEST11 TEST21.
       IS-FOR    I12.
 NI13  "$" FOLLOWED BY A NUMBER TRANSFORMS TO THE NUMBER FOLLOWED BY
          "DOLLARS".
       USED-FOR TEST5 TEST16 TEST19 TEST20.
       IS-FOR    I13.
 NI14  "CONSECUTIVE TO" TRANSFORMS TO "1 PLUS".
       USED-FOR TEST22.
       IS-FOR    I14.
 NI15  "LARGER THAN" TRANSFORMS TO "PLUS".
       USED-FOR TEST17.
       IS-FOR    I15.
 NI16  "PER CENT" PRECEDED BY A NUMBER TRANSFORMS TO NUMBER / 100.
       USED-FOR TEST4 TEST14 TEST24.
       IS-FOR    I16.
 NI17  "HOW MANY" TRANSFORMS TO "HOWM".
       USED-FOR TEST11 TEST12 TEST13.
       IS-FOR    I17.
       INTERP    D87.
 NI18  "THE SQUARE OF" TRANSFORMS TO "SQUARE".
       USED-FOR TEST4 TEST26.
       IS-FOR    I18.
 NI19  "MULTIPLIED BY" TRANSFORMS TO "TIMES".
       USED-FOR.
       IS-FOR    I19.
 NI20  "DIVIDED BY" TRANSFORMS TO "DIVBY".
       USED-FOR.
       IS-FOR    I20.
 NI21  "THE SUM OF" SHOULD BE REMOVED.
       USED-FOR TEST5 TEST7 TEST10 TEST16 TEST17 TEST20 TEST21 TEST22 TEST23
                   TEST24.
       IS-FOR    I21.
 NI22  "AND" TRANSFORMS TO "PLUSS" , IF "THE SUM OF" HAS PREVIOUSLY OCCURRED
          IN THE SAME CHUNK , AND IF THERE HAS BEEN NO OCCURRENCE OF "THE
          DIFFERENCE BETWEEN" WITHOUT A MATCHING "AND".
       USED-FOR TEST5 TEST7 TEST10 TEST16 TEST17 TEST20 TEST21 TEST22 TEST23
                   TEST24.
       IS-FOR    I21 I22.
 NI23  "THE DIFFERENCE BETWEEN" SHOULD BE REMOVED.
       USED-FOR TEST21 TEST24 TEST26.
       IS-FOR    I23.
 NI24  "AND" TRANSFORMS TO "MINUSS" , IF THERE HAS PREVIOUSLY OCCURRED IN THE
          SAME CHUNK "THE DIFFERENCE BETWEEN" , AND IF THERE HAS NOT ALREADY
          BEEN AN "AND" MATCHING THAT "THE DIFFERENCE BETWEEN".
       USED-FOR TEST21 TEST24 TEST26.
       IS-FOR    I23 I24.
 NI25  ", AND" TRANSFORMS TO "PERIOD".
```

```
      USED-FOR TEST4 TEST12 TEST16 TEST17 TEST19 TEST20 TEST25.
      FS-FOR   T25.
      INTER    T250.
N126  "." TRANSFORMS TO "PERIOD" , IF AN "IT" HAS BEEN DELETED PREVIOUSLY.
      USED-FOR TEST4 TEST6 TEST7 TEST8 TEST11 TEST12 TEST14 TEST15 TEST19
                  TEST25 TEST26 TEST27.
      FS-FOR   T26 T29.
      INTER    T260.
N1260 "." IS DELETED.
      USED-FOR TEST16.
      FS-FOR   T260.
N127  "?" TRANSFORMS TO "QMARK".
      USED-FOR TEST2 TEST4 TEST6 TEST7 TEST9 TEST11 TEST12 TEST13 TEST14
                  TEST18 TEST22.
      FS-FOR   T27.
N128  "." TRANSFORMS TO "PERIOD".
      USED-FOR ALL BUT TEST4 TEST12.
      FS-FOR   T28.
N129  "IT" SHOULD BE REMOVED.
      USED-FOR TEST4 TEST6 TEST7 TEST8 TEST11 TEST12 TEST14 TEST15 TEST19
                  TEST25 TEST26 TEST27.
      FS-FOR   T29.
N130  "TOTAL NUMBER" TRANSFORMS TO "NUMBER".
      USED-FOR TEST14.
      FS-FOR   T30.
N131  "EXCEEDS" TRANSFORMS TO "MINUS"
      USED-FOR TEST21.
      FS-FOR   T31.
N132  "BY" TRANSFORMS TO "IS" , IF THERE HAS BEEN AN "EXCEEDS" IN THE SAME
         CHUNK FOR WHICH THERE HAS NOT BEEN A MATCHING "BY".
      USED-FOR TEST21.
      FS-FOR   T31 T32.

ND

ND1   "PLUS" AND "LESSTHAN" ARE OPERATORS OF CLASS OP2.
      USED-FOR TEST1 TEST2 TEST9 TEST12 TEST19 TEST22 TEST23 TEST25.
      FS-FOR   D1 D3.
ND2   "TIMES" , "SQUARE" , "QUOTIENT" AND "OF" ARE OPERATORS OF CLASS OP1.
      USED-FOR ALL BUT TEST1 TEST3 TEST12.
      FS-FOR   D5 D7 D9 D11 M60 M62 M65 M75.
ND3   "SQUARED" , "**" , "MINUS" , "PER" , "PLUSS" AND "MINUSS" ARE
         OPERATORS OF CLASS OP0.
      USED-FOR TEST5 TEST6 TEST7 TEST9 TEST10 TEST15 TEST16 TEST17 TEST18
                  TEST20 TEST21 TEST22 TEST23 TEST24 TEST25 TEST26.
      FS-FOR   D13 D14 D15 D17 D18 D19.
ND4   "**" AND "PER" STAND FOR, RESPECTIVELY, OPERATORS EXPT AND QUOTIENT.
      USED-FOR TEST15 TEST16 TEST18.
      FS-FOR   D14 D17.
ND5   "HAS" , "GETS" , "HAVE" AND "WEIGHS" ARE VERBS.
      USED-FOR TEST4 TEST11 TEST13 TEST16 TEST20.
      FS-FOR   D21 D24 D27 D30.
ND6   "MARY" , "ANN" , "BILL" , "FATHER" AND "UNCLE" ARE PERSONS.
      USED-FOR TEST6 TEST9 TEST10 TEST11.
      FS-FOR   D41 D44 D47 D50 D53 D73 D83.
ND7   "HE" IS A PERSONAL PRONOUN IF IT OCCURS IN AN AGE PROBLEM.
      USED-FOR TEST9.
      FS-FOR   D55.
ND8   "HIS" IS A POSSESSIVE PRONOUN IF IT OCCURS IN AN AGE PROBLEM.
      USED-FOR TEST9.
      FS-FOR   D57.
ND9   "PEOPLE" , "FEET" , "YARDS" , "FATHOMS" , "INCHES" , "SPANS" , "MILES"
         , "GALLONS" , "HOURS" , "POUNDS" , "TONS" AND "DOLLARS" ARE PLURAL
         FORMS OF "PERSON" , "FOOT" , "YARD" , "FATHOM" , "INCH" , "SPAN" ,
         "MILE" , "GALLON" , "HOUR" , "POUND" , "TON" , AND "DOLLAR" ,
         RESPECTIVELY.
      USED-FOR TEST5 TEST7 TEST8 TEST12 TEST15 TEST16 TEST18 TEST19 TEST20
                  TEST22.
      FS-FOR   D61 D63 D65 D67 D69 D71 D72 D73 D75 D77 D78 D79.
ND10  "FATHOM" AND "SPAN" ARE SINGULAR FORMS OF "FATHOMS" AND "SPANS"
         RESPECTIVELY.
      USED-FOR TEST17.
      FS-FOR   D675 D715.
ND11  "WHAT" , "FIND" , "HOW" AND "HOW" ARE QWORDS.
      USED-FOR ALL.
      FS-FOR   D81 D83 D85 D87.
ND12  A PERIOD IS A DELIMITER.
      USED-FOR ALL.
      FS-FOR   D91.
ND13  WORDS THAT ARE OPERATORS OF CLASS OP2, OPERATORS OF CLASS OP1,
         OPERATORS OF CLASS OP0, VERBS, PERSONS, PERSONAL PRONOUNS,
         POSSESSIVE PRONOUNS, PLURALS, SINGULARS, QWORDS AND DELIMITERS
         SHOULD BE GIVEN DICTIONARY TAGS DENOTING THOSE THINGS.
      USED-FOR ALL.
      FS-FOR   D1 D3 D5 D7 D9 D11 D13 D14 D15 D17 D18 D19 D21 D24 D27 D30
                  D41 D44 D47 D50 D53 D55 D57 D61 D63 D65 D67 D675 D69 D71
                  D715 D72 D73 D75 D77 D78 D79 D81 D83 D85 D87 D91.
ND14  "IS" SHOULD BE GIVEN AN "IS" DICTIONARY TAG.
      USED-FOR ALL BUT TEST11.
      FS-FOR   D96.

NP

NP1   PRECEDENCES ARE ASSIGNED TO THE FOLLOWING IN DESCENDING ORDER : VERB,
         "IS" , OPERATOR OF CLASS OP2, "SQUARE" , OPERATOR OF CLASS OP1
         EXCEPT "SQUARE" , "SQUARED" , OPERATOR OF CLASS OP0 EXCEPT
         "SQUARED".
      USED-FOR ALL.
      FS-FOR   P1 P2 P3 P4 P5 P8 P9 M20 M30 M50 M55.
NP2   THE PRECEDENCE SCAN WORKS FIRST ON THE LEFTMOST UNSCANNED CHUNK IN
         TERMS OF RELATIVE POSITION IN THE PROBLEM.
      USED-FOR ALL.
      FS-FOR   P10 P15.
NP3   THE PRECEDENCE SCAN SCANS A CHUNK FROM LEFT TO RIGHT STARTING AT THE
         LEFT END OF THE CHUNK , GOING UP TO THE RIGHT END, RECORDING THE
         LEFTMOST OCCURRENCE OF THE HIGHEST PRECEDENCE IN THE CHUNK BY
         NOTING THE NEW MAXIMUM AT EACH POINT HAVING A PRECEDENCE HIGHER
         THAN THE PREVIOUS MAXIMUM.
      USED-FOR ALL.
      FS-FOR   S20 S25 S30 S40 P10 P15 P20 P23 P26 P27 P28 P29 P30 P35 P40
                  P45 P50 P65 P70.
NP4   IF NO WORD WITH A PRECEDENCE IS FOUND IN A CHUNK , IT IS A VARIABLE
         CHUNK.
      USED-FOR ALL.
      FS-FOR   P75

NM

NM1   A CHUNK WITH "IS" AS HIGHEST-PRECEDENCE WORD IS TREATED AS IF THE "IS"
         WERE THE OPERATOR "EQUAL".
      USED-FOR ALL.
      FS-FOR   M10.
NM2   A CHUNK OF THE FORM (S-1 V-1 S-2 " AS MANY" S-3 " AS" S-4 V-2 S-5)
         WHERE V-1 AND V-2 ARE TWO VERBS, WHERE THE S'S ARE SEGMENTS OF
         STRINGS, AND WHERE THE FIRST VERB IS THE HIGHEST PRECEDENCE ELEMENT
         OF THE CHUNK , TRANSFORMS TO A NEW CHUNK OF THE FORM
         (S-2 " THE NUMBER OF" S-3 S-1 V-1 " IS THE NUMBER OF" S-5 S-4 V-2).
      USED-FOR TEST11 TEST20.
      FS-FOR   M20.
NM3   A CHUNK OF THE FORM (S-1 V-1 " AS MANY" S-3 " AS" S-4 V-2 S-5) WHERE
         V-1 AND V-2 STAND FOR TWO VERBS, WHERE THE S'S ARE SEGMENTS OF
         STRINGS, AND WHERE V-1 IS THE HIGHEST-PRECEDENCE ELEMENT OF THE
         CHUNK , TRANSFORMS TO A NEW CHUNK OF THE FORM
         (" THE NUMBER OF" S-3 S-1 V-1 " IS THE NUMBER OF" S-5 S-4 V-2).
      USED-FOR.
      FS-FOR   M30.
NM4   A VERB WHICH HAS TAKEN PART IN A TRANSFORMATION NO LONGER HAS ITS
         PRECEDENCE.
      USED-FOR TEST11 TEST16 TEST20.
      FS-FOR   M20 M30 M50 M55.
NM5   A VERB WHICH IS THE HIGHEST-PRECEDENCE ELEMENT OF A CHUNK , AND WHICH
         CANNOT TAKE PART IN ANY TRANSFORMATIONS, LOSES ITS PRECEDENCE, AND
         THE CONTAINING CHUNK MUST HAVE ANOTHER PRECEDENCE SCAN.
      USED-FOR TEST4 TEST13.
      FS-FOR   M40.
NM6   A CHUNK OF THE FORM (S-1 V N X S-2) , WHERE THE S'S ARE SEGMENTS OF
         STRINGS, WHERE X IS A SINGLE WORD, WHERE N IS A NUMBER, AND WHERE V
         IS A VERB AND THE HIGHEST-PRECEDENCE ELEMENT OF THE CHUNK,
         TRANSFORMS TO A NEW CHUNK OF THE FORM
         (" THE NUMBER OF" X S-1 V " IS" N S-2).
      USED-FOR.
      FS-FOR   M50.
      INTER    M60.
NM7   A CHUNK OF THE FORM (S V N X) , WHERE S IS A STRING SEGMENT, WHERE X
         IS A SINGLE WORD, WHERE N IS A NUMBER, AND WHERE V IS A VERB AND IS
         THE HIGHEST-PRECEDENCE ELEMENT OF THE CHUNK , TRANSFORMS TO A NEW
         CHUNK OF THE FORM (" THE NUMBER OF" X S V " IS" N).
      USED-FOR TEST11 TEST16.
      FS-FOR   M55.
      INTER    M60.
NM8   A NEW CHUNK THAT IS THE RESULT OF A TRANSFORMATION MUST UNDERGO THE
         PRECEDENCE SCAN.
      USED-FOR TEST11 TEST16 TEST20.
      FS-FOR   M20 M30 M50 M55.
NM9   A PLURAL ELEMENT TRANSFORMS TO "TIMES" FOLLOWED BY THE SINGULAR FORM
         OF THE PLURAL ELEMENT, IF THE PLURAL IS PRECEDED BY A NUMBER.
```

```
     USED-FOR TEST5 TEST7 TEST8 TEST12 TEST15 TEST16 TEST18 TEST19 TEST20
               TEST22.
     ES-FOR    M60.
NM10 IF A SINGULAR FORM OF A WORD IS PRECEDED BY THE NUMBER 1 , THERE
        SHOULD BE PLACED BETWEEN THEM THE OPERATOR "TIMES".
     USED-FOR TEST12.
     ES-FOR    M62.
NM11 IF THE OPERATOR "OF" IS PRECEDED BY A NUMBER, IT SHOULD BE CHANGED TO
        "TIMES" , OTHERWISE IT IS NOT AN OPERATOR.
     USED-FOR ALL BUT TEST1 TEST13 TEST6 TEST10 TEST12 TEST17 TEST21 TEST23
               TEST25.
     ES-FOR    M65 M75.
     INTER     P5.
NM12 WHEN A CHUNK IS TRANSFORMED AS A RESULT OF HAVING A VERB AS ITS
        HIGHEST PRECEDENCE ELEMENT, IT IS A NEW CHUNK ; ALL NEW ELEMENTS
        SHOULD BE NOTED AS BELONGING IN THE CHUNK , AND A NEW PRECEDENCE
        SCAN SHOULD BE DONE.
     USED-FOR TEST11 TEST16 TEST20.
     ES-FOR    M20 M30 M50 M55.
NM13 WHEN NEW WORDS ARE ADDED TO A CHUNK AFTER THE INITIAL SCAN, THEY MUST
        BE NOTED AS BEING IN THE CHUNK , AND RELEVANT DICTIONARY TAGS MUST
        BE ADDED.
     USED-FOR TEST4 TEST5 TEST7 TEST8 TEST11 TEST12 TEST15 TEST16 TEST18
               TEST19 TEST20 TEST25 TEST22.
     ES-FOR    M20 M30 M50 M55 M60 M62 C15 C17 C50 C52 P2 P9.

. . .  NC

NC1  THE PHRASES "IS MULTIPLIED BY" , "IS DIVIDED BY" , AND "IS INCREASED
        BY" ARE OPERATORS STANDING RESPECTIVELY FOR "TIMES" , "QUOTIENT" ,
        AND "PLUS".
     USED-FOR TEST19.
     ES-FOR    C2 C5 C8.
     INTER     [illegible]
NC2  CHUNKS WITH OPERATORS STARTING WITH "IS" , EXCEPT THE OPERATOR
        CONSISTING OF "IS" ALONE, WILL BECOME EXPRESSIONS TO WHICH FUTURE
        PHRASES CONTAINING "THIS" WILL REFER; WHEN FORMED, THEY CAN BE
        RECOGNIZED AS BEING TOP LEVEL EXPRESSIONS WITH OPERATOR NOT
        "EQUAL".
     USED-FOR ALL.
     ES-FOR    C2 C5 C8 C70 C80 C85.
NC3  OPERATORS EXCEPT "SQUARE" AND "SQUARED" AND OPERATORS STARTING WITH
        "IS" ARE CARRIED OVER DIRECTLY INTO RESULTING EXPRESSIONS FOR THE
        CHUNKS THEY OCCUR IN.
     USED-FOR ALL BUT TEST13.
     ES-FOR    C10 C75 C56.
NC4  A CHUNK WITH A HIGHEST PRECEDENCE OPERATOR MARKED, EXCEPT "SQUARE" AND
        "SQUARED" , IS SPLIT INTO TWO NEW CHUNKS, WITH THE LEFT END OF THE
        LEFT CHUNK THE SAME AS THE ORIGINAL, RIGHT END OF THE LEFT CHUNK
        THE WORD DIRECTLY TO THE LEFT OF THE PHRASE REPRESENTING THE
        OPERATOR, LEFT END OF THE RIGHT CHUNK DIRECTLY TO THE RIGHT OF THE
        PHRASE REPRESENTING THE OPERATOR, AND RIGHT END OF THE RIGHT CHUNK
        AT THE RIGHT END OF THE ORIGINAL CHUNK.
     USED-FOR ALL.
     ES-FOR    M10 C2 C5 C8 C10 C12 C22 C25 C52 C55 C60.
NC5  WHEN A CHUNK IS SPLIT, THE OPERATOR AT THE SPLITTING POINT BECOMES THE
        OPERATOR OF THE RESULTING EXPRESSION, WITH THE EXPRESSION OF THE
        LEFT HALF OF THE CHUNK BECOMING THE LEFT OPERAND OF THE EXPRESSION
        AND WITH THE EXPRESSION OF THE RIGHT HALF BECOMING THE RIGHT
        OPERAND.
     USED-FOR ALL.
     ES-FOR    M10 C2 C5 C8 C10 C22 C25 C55 C60 C70 C75 C80.
NC6  WHEN "SQUARE" IS MARKED AS HIGHEST-PRECEDENCE OPERATOR OF A CHUNK , IT
        IS A UNARY EXPONENTIAL OPERATOR WHOSE OPERAND IS THE PORTION OF THE
        CONTAINING CHUNK TO THE RIGHT OF THE "SQUARE".
     USED-FOR TEST4 TEST26.
     ES-FOR    C15 C17.
NC7  WHEN "SQUARED" IS MARKED AS HIGHEST-PRECEDENCE OPERATOR OF A CHUNK , IT
        IS A UNARY EXPONENTIAL OPERATOR WHOSE OPERAND IS THE PORTION OF THE
        CONTAINING CHUNK TO THE LEFT OF THE "SQUARED".
     USED-FOR.
     ES-FOR    C50 C52.
NC8  THE EXPRESSION REPRESENTED BY A UNARY EXPONENTIAL OPERATOR BECOMES AN
        OPERAND IN THE EXPRESSION OF THE REMAINDER OF THE CONTAINING CHUNK;
        IF THE REMAINDER IS NOT EMPTY, A DUMMY SHOULD REPLACE THE UNARY
        OPERATOR AND OPERAND IN THAT REMAINDER TO ALLOW THE TWO EXPRESSIONS
        TO BE FORMED INDEPENDENTLY.
     USED-FOR TEST4.
     ES-FOR    C15 C50.
     INTER     C17 C52.
NC9  A UNARY EXPONENTIAL OPERATOR HAS "EXPT" AS ITS OPERATOR WITH ITS
        OPERAND SERVING AS THE FIRST OPERAND OF "EXPT" AND WITH "2 "AS THE
        SECOND OPERAND OF "EXPT".
```

```
     USED-FOR TEST4 TEST26.
     ES-FOR    C15 C17 C50 C52.
NC10 WHEN A UNARY OPERATOR CHUNK IS SEPARATED FROM ITS CONTAINING CHUNK,
        THE CONTAINING CHUNK NEEDS A PRECEDENCE SCAN; THE UNARY EXPRESSION
        SHOULD BE FORMED BY ADDING THE OPERATOR AND SECOND OPERAND,
        RENAMED, AND THEN SPLIT AS IF A BINARY OPERATOR EXPRESSION.
     USED-FOR TEST4 TEST26.
     ES-FOR    C15 C17 C20 C22 C50 C52.
NC11 AN EXPRESSION IS A TREE STRUCTURE OF THE FORM (a b c) WHERE a IS THE
        OPERATOR, b IS THE TREE EXPRESSION FOR THE LEFT OPERAND, AND c IS
        THE SAME FOR THE RIGHT OPERAND.
     USED-FOR ALL.
     ES-FOR    C70 C75 C78.
NC12 AN EXPRESSION WITH OPERATOR "EQUAL" AT THE TOP LEVEL IS AN EQUATION.
     USED-FOR ALL.
     ES-FOR    M10 C90.
NC14 THE EXPRESSION FOR THE LEFT OPERAND OF THE OPERATOR "EQUAL" IS
        SOMETHING TO WHICH A FUTURE PHRASE CONTAINING "THIS" MAY REFER.
     USED-FOR ALL.
     ES-FOR    C75 C80 C85.
     INTER     C70.
NC15 WHEN A CHUNK IS SPLIT, THE PRECEDENCE SCAN ON THE NEW CHUNKS SHOULD BE
        DONE ON THE LEFT OPERAND CHUNK BEFORE THE RIGHT OPERAND CHUNK; ALSO
        BOTH CHUNKS SHOULD BE FORMED BY RENAMING BEFORE ANY PRECEDENCE
        SCANNING IS DONE.
     USED-FOR ALL.
     ES-FOR    C60.
NC16 EXPRESSIONS TO WHICH A FUTURE "THIS" MAY REFER SHOULD NOT BE NOTED
        UNTIL AFTER THE EXPRESSION IS FORMED, SINCE THE COMPONENTS OF THE
        EXPRESSION MAY CONTAIN A "THIS" WHICH REFERS BACK TO A PREVIOUS
        EXPRESSION.
     USED-FOR.
     ES-FOR .
     INTER     C2 C5 C8.
NC17 AN EXPRESSION IS FORMED AND REQUIRES NO FURTHER PROCESSING ON SUBPARTS
        WHEN IT IS RECOGNIZED AS A VARIABLE CHUNK OR WHEN IT IS COMPOSED OF
        ENTITIES THAT ARE KNOWN TO BE SO FORMED.
     USED-FOR ALL.
     ES-FOR    P25 C70 C75 C78.

. . .  NR . . .

NR1  A NEW CHUNK IS FORMED BY RENAMING A PART OF THE OLD CHUNK WHICH IS ITS
        PARENT CHUNK.
     USED-FOR ALL.
     ES-FOR    M20 M30 M50 M55 C20 C60 P2 P4 P6 P7 P8 P9.
NR2  A NEW CHUNK CAN (AND SHOULD) UNDERGO PRECEDENCE SCANNING, BUT ONLY
        AFTER IT HAS BEEN FORMED BY RENAMING THE CHUNK OF WHICH IT WAS A
        PART.
     USED-FOR ALL.
     ES-FOR    C60 P4 P8 P9.

. . .  NV . . .

NV1  THREE CHECKS ARE PERFORMED IN TURN ON A NEW VARIABLE CHUNK TO
        DETERMINE WHETHER IT REFERS TO AN EXPRESSION ALREADY KNOWN, OR
        WHETHER IT IS A NEW UNIQUE PROBLEM VARIABLE; IF ANY CHECK SUCCEEDS,
        THE VARIABLE REFERS TO A PREVIOUS ONE, AND THE OTHERS NEED NOT BE
        DONE.
     USED-FOR ALL.
     ES-FOR    P25 V5 V15 V50 V65 V80 V85 V90.
NV2  THE FIRST CHECK TO BE MADE ON A NEW VARIABLE CHUNK IS WHETHER IT IS
        THE DUMMY CHUNK FOR A UNARY OPERATOR EXPRESSION; IF SO, IT REFERS
        TO THAT UNARY OPERATOR EXPRESSION, AND IS NOT ITSELF A VARIABLE.
     USED-FOR ALL.
     ES-FOR    V5 V10.
     INTER     P25.
NV3  THE SECOND CHECK ON A NEW VARIABLE CHUNK IS FOR THE WORD "THIS" IN THE
        CHUNK; IF SO, IT REFERS TO THE MOST RECENT EXPRESSION WHICH WAS
        NOTED AS A POSSIBLE REFERRENT FOR A CHUNK CONTAINING "THIS".
     USED-FOR ALL.
     ES-FOR    V5 V15 V20 V21.
NV4  THE THIRD CHECK ON A NEW VARIABLE CHUNK IS WHETHER IT MATCHES ANY OF
        THE VARIABLES ALREADY SEEN IN THE PROBLEM.
     USED-FOR ALL.
     ES-FOR    V20 V21 V23 V24 V25 V26 V55.
NV5  IF ANY OF THE PREVIOUSLY-SEEN VARIABLES IS KNOWN TO MATCH AN EARLIER
        VARIABLE, IT NEED NOT BE MATCHED AGAINST THE NEW VARIABLE CHUNK.
     USED-FOR ALL.
     ES-FOR    V20 V21 V23 V24 V25.
NV6  THE MATCH IS CARRIED OUT FROM LEFT TO RIGHT, COMPARING THE TWO CHUNKS
        WORD FOR WORD.
     USED-FOR ALL.
```

STUDENT

```
      IS-FOR   V26 V30 V31 V32 V33 V33R V34.
NV7   THE MATCH SUCCEEDS IF THE END OF THE NEW CHUNK IS REACHED BEFORE THERE
         IS A POSITION IN THE TWO CHUNKS WHICH DOESN'T MATCH.
      USED-FOR ALL.
      IS-FOR   V35 V36R V37 V40.
      INTER    V30 V31 V34.
NV8   IF THE MATCH SUCCEEDS, THE NEW CHUNK HAS THE SAME EXPRESSION AS THE
         CHUNK TO WHICH IT MATCHES.
      USED-FOR ALL.
      IS-FOR   V50.
NV9   TWO WORDS MATCH IF THEY ARE IDENTICAL.
      USED-FOR ALL.
      IS-FOR   V30 V40.
NV10  TWO WORDS MATCH IF THE WORD FROM THE NEW CHUNK IS "THE" AND THE WORD
         FROM THE OTHER CHUNK IS "A".
      USED-FOR TEST12 TEST13 TEST17 TEST16 TEST20.
      IS-FOR   V31.
NV11  TWO WORDS MATCH IF THE WORD FROM THE NEW CHUNK IS "THEY" AND THE WORD
         FROM THE OTHER CHUNK IS "THE" ; IN THIS CASE THE MATCH MUST SKIP
         OVER THE WORD AFTER THE "THE" ; THIS NEVER OCCURS AT THE RIGHT END
         OF A CHUNK.
      USED-FOR TEST13.
      IS-FOR   V32.
NV12  IF ONE OF THE CHUNKS HAS "THE" FOLLOWED BY A WORD WHICH IS IDENTICAL
         TO THE WORD IN THE OTHER CHUNK CORRESPONDING TO THE "THE" , THE TWO
         CHUNKS MATCH AT THAT POSITION.
      USED-FOR TEST15 TEST17 TEST21 TEST22 TEST23 TEST24.
      IS-FOR   V31 V33R.
NV13  A WORD IN THE NEW CHUNK MATCHES "THE NUMBER OF" FOLLOWED BY THE PLURAL
         FORM OF THE WORD, IN THE OTHER CHUNK.
      USED-FOR TEST16.
      IS-FOR   V34.
NV14  "FIRST" IN THE NEW CHUNK MATCHES "ONE" IN THE OLD CHUNK , PROVIDED THAT
         THE WORD FOLLOWING EACH OF THEM IS "NUMBER" , AND THAT THE "NUMBER"
       . IS THE LAST WORD IN THE NEW CHUNK.
      USED-FOR TEST17.
      IS-FOR   V36.
NV15  "FIRST NUMBER" IN THE OLD CHUNK MATCHES "ONE OF THE NUMBERS" IN THE
         NEW CHUNK , IF THE "NUMBERS" IS THE LAST WORD IN THE NEW CHUNK.
      USED-FOR TEST17.
      IS-FOR   V36R.
NV16  "SECOND NUMBER" IN THE OLD CHUNK MATCHES "OTHER NUMBER" IN THE NEW
         CHUNK , IF THE "NUMBER" IS THE LAST WORD IN THE NEW CHUNK.
      USED-FOR TEST17 TEST22.
      IS-FOR   V37.
NV17  IF THE NEW VARIABLE DOES NOT MATCH SUCCESSFULLY TO ANY OF THE OTHER
         VARIABLES, IT IS A NEW UNIQUE PROBLEM VARIABLE.
      USED-FOR ALL.
      IS-FOR   V60.
NV18  IF THE NEW VARIABLE SUCCEEDS IN MATCHING ONE OF THE OLD VARIABLES, ALL
         OTHER MATCHINGS ON THE NEW VARIABLE CAN BE TERMINATED.
      USED-FOR ALL.
      IS-FOR   V50 V60 V61.

[illegible] NF [illegible]

NF1   A SENTENCE BEGINNING WITH "WHAT ARE" WILL HAVE FV CHUNKS STARTING
         AFTER THE "ARE" AND SEPARATED BY "AND" , AND IT WILL END WITH
         "QMARK".
      USED-FOR TEST2 TEST9.
      IS-FOR   E5 E15 E20.
NF2   "AND" , "QMARK" AND "PERIOD" ARE NOT PART OF FV CHUNKS.
      USED-FOR ALL.
      IS-FOR   E5 E15 E20 E35 E70 E75.
      INTER    S15.
NF3   AFTER THE RIGHT END OF AN FV CHUNK HAS BEEN REACHED, THE CHUNK SHOULD
         BE PROCESSED AS A NEW VARIABLE CHUNK TO DETERMINE ITS REFERENT.
      USED-FOR ALL.
      IS-FOR   E15 E20 E35 E70 E75.
NF4   A SENTENCE BEGINNING WITH "WHAT IS" HAS ONE FV CHUNK STARTING AFTER
         THE "IS" AND ENDING BEFORE THE "QMARK" WHICH ENDS THE SENTENCE.
      USED-FOR TEST4 TEST6 TEST7 TEST11 TEST12 TEST13 TEST14 TEST18 TEST27.
      IS-FOR   E25 E35.
NF5   A SENTENCE WHOSE BEGINNING IS OF THE FORM (" HOWM" X " EQUALS") ,
         WHERE X IS ANY WORD, HAS AN FV CHUNK STARTING AFTER THE "EQUALS"
         AND ENDING BEFORE THE "QMARK" WHICH ENDS THE SENTENCE AND HAS X AS
         AN ANSWER UNIT.
      USED-FOR TEST4 TEST6 TEST7 TEST11 TEST12 TEST13 TEST14 TEST18 TEST27.
      IS-FOR   E35 E40.
NF5A  AN ANSWER UNIT IS A PLURAL FORM OF A WORD.
      USED-FOR TEST12.
      IS-FOR   E3.
      INTER    E40.
NF6   A SENTENCE WHOSE BEGINNING IS OF THE FORM (" HOWM" X " DO" Y) , WHERE
         X IS ANY WORD, AND WHERE Y IS ANY WORD, HAS AN FV CHUNK WHICH
         STARTS OUT (" THE NUMBER OF" X Y) AND CONTINUES AFTER THE Y UNTIL
         THE "QMARK" WHICH ENDS THE SENTENCE.
      USED-FOR TEST4 TEST6 TEST7 TEST11 TEST12 TEST13 TEST14 TEST18 TEST27.
      IS-FOR   E35 E45 E48.
NF7   A SENTENCE OF THE FORM (" HOWM" X " DOES" S " HAVE QMARK") , WHERE X
         IS ANY WORD, AND WHERE S IS A STRING OF WORDS, HAS AN FV CHUNK OF
         THE FORM (" THE NUMBER OF" X S " HAS") FOLLOWED BY "QMARK".
      USED-FOR TEST4 TEST6 TEST7 TEST11 TEST12 TEST13 TEST14 TEST18 TEST27.
      IS-FOR   E35 E50 E52 E55.
NF8   A SENTENCE WHICH STARTS WITH "FIND" HAS FV CHUNKS STARTING AFTER THE
         "FIND" AND SEPARATED BY "AND" , AND IT ENDS WITH "PERIOD".
      USED-FOR TEST1 TEST3 TEST5 TEST8 TEST10 TEST15 TEST16 TEST17 TEST19
               TEST20 TEST21 TEST22 TEST23 TEST24 TEST25 TEST26.
      IS-FOR   E60 E70 E75.
      INTER    S15.
NF9   WHEN CHANGING A STRING ACCORDING TO FV TRANSFORMATIONS, IT IS NOT
         NECESSARY TO REARRANGE SCAN POINTERS.
      USED-FOR.
      IS-FOR .
      INTER    E5 E25 E40 E45 E50 E60.

[illegible] NA [illegible]

NA1   A PROBLEM IS AN AGE PROBLEM IF IT HAS THE PHRASE "AS OLD AS" , "AGE" ,
         OR "YEARS OLD".
      USED-FOR TEST6 TEST9 TEST10.
      IS-FOR   A1 A2 A3.
NA2   THE PHRASES "AS OLD AS" AND "YEARS OLD" ARE DELETED, DURING THE
         TRANSFORMATION PROCESS.
      USED-FOR TEST6 TEST10.
      IS-FOR   A11 A12.
NA3   "WILL BE WHEN" TRANSFORMS TO "IN X YEARS , IN X YEARS" , WHERE X IS AN
         ARBITRARY UNIQUE SYMBOL, IF THE PROBLEM IS AN AGE PROBLEM.
      USED-FOR TEST6 TEST9 TEST10.
      IS-FOR   A15 A30.
      INTER    A19.
NA4   "WAS WHEN" TRANSFORMS TO "X YEARS AGO , X YEARS AGO" , WHERE X IS AN
         ARBITRARY UNIQUE SYMBOL, IF THE PROBLEM IS AN AGE PROBLEM.
      USED-FOR TEST6 TEST9 TEST10.
      IS-FOR   A17 A30.
      INTER    A18.
NA5   "WAS" AND "WILL BE" TRANSFORM TO "IS" , IN AGE PROBLEMS.
      USED-FOR TEST6 TEST9 TEST10.
      IS-FOR   A18 A19.
NA6   "IS NOW" TRANSFORMS TO "S AGE NOW" , IN AGE PROBLEMS.
      USED-FOR TEST6 TEST9.
      IS-FOR   A20.
NA7   AT THE BEGINNING OF CHUNKS IN AGE PROBLEMS, IF THERE IS "IN X YEARS" ,
         "X YEARS AGO" , OR "X YEARS FROM NOW" , WHERE X IS ANY SINGLE WORD
         IN EACH CASE, THAT PHRASE IS DELETED FROM THE CHUNK BUT IS SAVED AS
         AN AGE OPERATOR FOR UNMODIFIED AGE VARIABLES IN THE CHUNK.
      USED-FOR TEST6 TEST9 TEST10.
      IS-FOR   A24 A26 A28 A41 A42 A43.
NA8   "AGE" WHICH IS NOT FOLLOWED BY A PHRASE WHICH TRANSFORMS TO "PLUSS" OR
         "MINUSS" , OR BY A PHRASE WHICH TRANSFORMS TO SUCH A PHRASE, IS
         UNMODIFIED, AND NEEDS TO BE TRANSFORMED SO THAT IT IS FOLLOWED BY
         THE AGE OPERATOR FOR THE CURRENT CHUNK.
      USED-FOR TEST6 TEST9 TEST10.
      IS-FOR   A30.
NA9   "AGE IN X YEARS" , AND "AGE X YEARS FROM NOW" , WHERE X STANDS FOR A
         SINGLE WORD, TRANSFORM TO "AGE PLUSS X".
      USED-FOR TEST6 TEST9 TEST10.
      IS-FOR   A31 A32 A30.
NA10  "AGE X YEARS AGO" , WHERE X STANDS FOR A SINGLE WORD, TRANSFORMS TO
         "AGE MINUSS X".
      USED-FOR TEST6 TEST9 TEST10.
      IS-FOR   A34 A30.
NA11  "AGE NOW" TRANSFORMS TO "AGE".
      USED-FOR TEST6 TEST9 TEST10.
      IS-FOR   A35 A30.
NA12  "THEIR AGES" , IN AN AGE PROBLEM, TRANSFORMS TO A LIST OF ALL THE
         DISTINCT AGE VARIABLES SEEN SO FAR IN THE PROBLEM, SEPARATED BY
         "AND" , IN THEIR ORDER OF OCCURRENCE IN THE PROBLEM.
      USED-FOR TEST6 TEST9 TEST10.
      IS-FOR   A50 A51 A56 A57 A59 A62 A63 A631.
NA13  IF A PERSON, IN AN AGE PROBLEM, IS NOT FOLLOWED BY "S" , THEN THE
         PHRASE "S AGE" SHOULD BE INSERTED AFTER IT.
      USED-FOR TEST6 TEST9 TEST10.
      IS-FOR   A66 A68.
      INTER    A20.
NA14  AN AGE VARIABLE, IN AN AGE PROBLEM, CONSISTS OF A PERSON NOT PRECEDED
```

```
       BY "S" AND FOLLOWED BY "S SEG AGE" , WHERE SEG IS A STRING OF 0 OR
       MORE REPETITIONS OF THE FORM "X S" WHERE X IS A PERSON.
    USED-FOR TEST6 TEST9 TEST10.
    IS-FOR    A62 A63 A62 A67 A68 A69 A73 A75 A83 A85.
NA15  A NEW AGE VARIABLE, IN AN AGE PROBLEM, IS DISTINCT FROM PREVIOUS ONES
       IF IT IS NOT EXACTLY THE SAME AS ANY OTHER.
    USED-FOR TEST6 TEST9 TEST10.
    IS-FOR    A61 A631 A64 A67 A68 A69.
NA16  A PERSONAL PRONOUN IS REPLACED, IN AGE PROBLEMS, BY THE FIRST AGE
       VARIABLE SEEN IN THE PROBLEM.
    USED-FOR TEST9.
    IS-FOR    A63 A67 A75.
NA17  A POSSESSIVE PRONOUN, IN AGE PROBLEMS, IS REPLACED BY THE FIRST AGE
       VARIABLE SEEN IN THE PROBLEM, EXCEPT FOR THE FINAL WORD "AGE".
    USED-FOR TEST9.
    IS-FOR    A81 A83 A85.

    NB

NB1   THE ANSWER BUILDING PROCESS CONSISTS OF CONSTRUCTING A LIST OF THE
       VARIABLES OF EQ'S IN THE PROBLEM, CHECKING FOR NEED OF AN ANSWER
       UNIT, AND COLLECTING THE EXTERNAL REPRESENTATIONS OF THE VARIABLES
       OF THE PROBLEM.
    USED-FOR ALL.
    IS-FOR    B1 B2 B3.
NB2   THE LIST OF EQ'S SHOULD BE COLLECTED SO THAT IT WILL HAVE THE
       VARIABLES IN THE ORDER IN WHICH THEY WERE ENCOUNTERED IN THE
       INITIAL SCAN.
    USED-FOR ALL.
    IS-FOR    B2.
NB3   THE EXTERNAL REPRESENTATIONS OF VARIABLES SHOULD BE COLLECTED IN THE
       ORDER OF APPEARANCE OF THE STRINGS IN THE PROBLEM; IN CASE OF
       STRINGS REPRESENTING THE SAME VARIABLE, ONLY THE EXTERNAL
       REPRESENTATION OF THE FIRST ONE SEEN IN THE PROBLEM SHOULD BE
       COLLECTED.
    USED-FOR ALL.
    IS-FOR    B5.

    NI

NI1   PROBLEM SPACE SIZES CAN BE SUMMARIZED BY ESTIMATING THE NUMBER OF
       EQUATIONS, EQ'S, OPERATORS, AND VARIABLES, AND BY COUNTING
       EQUATIONS, EQ'S, OPERATORS (NOTED AND PLACED IN EXPRESSIONS) , AND
       VARIABLES (NOTED, DISTINCT, AND PLACED IN EXPRESSIONS).
    USED-FOR ALL.
    IS-FOR    S60 D1 D3 D5 D7 D9 D11 D13 D14 D15 D17 D18 D19 D10 D65 E2 C5
              C8 C60 U50 U60 F15 F25 13 111 113 115 117 121 131 133 141
              143.
NI2   THE NUMBER OF EQUATIONS IN THE PART OF THE PROBLEM AS YET UNSCANNED IS
       BOUNDED ABOVE BY (L - 3) / 4 , WHERE L IS THE LENGTH AS YET
       UNSCANNED.
    USED-FOR ALL.
    IS-FOR    11 15.
NI3   THE NUMBER OF EQ'S IN THE PART OF THE PROBLEM AS YET UNSCANNED IS
       BOUNDED ABOVE BY (L - 1) / 4 , WHERE L IS THE LENGTH AS YET
       UNSCANNED.
    USED-FOR ALL.
    IS-FOR    11 15.
NI4   THE NUMBER OF OPERATORS IN THE PART OF THE PROBLEM AS YET UNSCANNED IS
       BOUNDED ABOVE BY (L - 3) / 2 , WHERE L IS THE LENGTH AS YET
       UNSCANNED.
    USED-FOR ALL.
    IS-FOR    11 15.
NI5   THE NUMBER OF VARIABLES IN THE PART OF THE PROBLEM AS YET UNSCANNED IS
       BOUNDED ABOVE BY (L - 3) / 2 , WHERE L IS THE LENGTH AS YET
       UNSCANNED.
    USED-FOR ALL.
    IS-FOR    11 15.
NI6   THE NUMBER OF VARIABLES IN THE PROBLEM IS TWICE THE NUMBER OF
       EQUATIONS PLUS THE NUMBER OF OPERATORS.
    USED-FOR ALL.
    IS-FOR    111 113 121.
NI7   THE LENGTH OF THE PART OF THE PROBLEM AS YET UNSCANNED CHANGES EACH
       TIME A NEW OPERATOR, EQUATION, OR PERIOD IS SCANNED, AND IT CHANGES
       BY THE NUMBER OF WORDS SCANNED SINCE THE LAST CHANGE OR SINCE THE
       BEGINNING OF THE PROBLEM.
    USED-FOR ALL.
    IS-FOR    S13 S40 T50 T52 D11 D36 M60 M62 17 111 113 115.
NI8   WHEN THE FIRST EQ OCCURS, THE ESTIMATES BASED ON LENGTH AS YET
       UNSCANNED BECOME 0.
    USED-FOR ALL.
    IS-FOR    S15 S65 141.
NI9   THE LENGTH OF THE PART OF THE PROBLEM AS YET UNSCANNED MUST BE CHANGED
       WHEN WORDS ARE ADDED OR REMOVED BY A TRANSFORMATION, IE, WHEN THE
       OLD STRING IS NOT THE SAME LENGTH AS THE NEW.
    USED-FOR ALL BUT TEST1 TEST3.
    IS-FOR    11 12 13 14 15 16 18 19 19E 110 111 112 115 116 117 118 119
              120 121 123 125 1260 129 130 H60 H62 A11 A12 A15 A17 A19
              A20 A24 A26 A28 A31 A32 A34 A35 A41 A59 A61 A75 A85.
NI10  CHANGES RELEVANT TO SPACE SIZE ELEMENTS SHOULD BE PROCESSED BEFORE
       OTHER CHANGES.
    USED-FOR ALL.
    IS-FOR    S13 S60 11 12 13 14 15 16 18 19 19E 110 111 112 115 116 117
              118 119 120 121 123 125 1260 129 130 D1 D3 D5 D7 D9 D11
              D13 D14 D15 D17 D18 D19 D31 D96 M10 M60 M62 C2 C5 C8 C60
              U50 U60 F15 F25 A11 A12 A15 A17 A19 A20 A24 A26 A28 A31
              A32 A34 A35 A41 A59 A61 A75 A85.

    Q

Q1    WHEN A NEW ENTITY IS INTRODUCED, COUNTERS AND RECORDERS OF INFORMATION
       ABOUT THAT ENTITY SHOULD BE INITIALIZED.
    USED-FOR ALL.
    IS-FOR    S10 S40 S60 S65 M20 M30 M50 M55 U5 F15 F25 F80.
       AT THE BEGINNING OF A PROBLEM, COUNTERS AND RECORDERS OF INFORMATION
Q2     THAT ARE GLOBAL TO THE PROBLEM SHOULD BE INITIALIZED.
    USED-FOR ALL.
    IS-FOR    S10 B1 B2 B3 11.
       FOR STORAGE EFFICIENCY, PROGRAM SEGMENTS THAT RESPOND TO SCAN SIGNALS
Q3     OF THE "COMPLETION" TYPE SHOULD ALSO REMOVE THE CORRESPONDING
       "INITIATE" TYPE, AS WELL AS REMOVING THE USED "COMPLETION" SIGNAL,
       IF IT IS POSSIBLE THAT NO PROGRAM SEGMENT RESPONDS TO THE INITIATE
       SIGNAL.
    USED-FOR ALL.
    IS-FOR    S13 S15 S16 S17 S18 S40.
       THE PROCESS OF SCANNING INVOLVES MOVING A SCAN POSITION FROM AN OLD
Q4     POSITION TO A NEW ONE.
    USED-FOR ALL.
    IS-FOR    S13 S15 P20 P23 P26.
       APPLYING A NUMBER OF FUNCTIONS IN TURN MEANS TO APPLY THE FIRST, AND
Q5     WHEN THAT IS DONE, APPLY THE SECOND, AND SO ON.
    USED-FOR ALL.
    IS-FOR    S10 S13 S15 S16 S17 S18 S35 S40 U5.
       WHEN A VALUE OF A COUNTER IS CHANGED, THE OLD VALUE SHOULD BE REMOVED.
Q6  USED-FOR ALL.
    IS-FOR    S13 S40 S65 C60 U5 U50 U55 F80 A64 13 15 111 113 115 121 131
              133 141 143.
       IF NO FURTHER TRANSFORMATIONS APPLY TO THE RESULTS OF A GIVEN ONE,
Q7     DELETE THE APPLY-TRANSFORMATION SIGNAL.
    USED-FOR ALL.
    IS-FOR    11 12 12E 13 14 15 16 17 18 19 19E 110 111 112 113 114 115
              116 117 118 119 120 121 122 123 124 125 126 1260 127 128
              129 130 131 132 D1 D3 D5 D7 D9 D11 D13 D14 D15 D17 D18 D19
              D21 D24 D27 D30 D31 D44 D47 D50 D53 D55 D57 D61 D63 D65
              D67 D675 D69 D71 D715 D77 D73 D75 D77 D78 D79 D81 D83 D85
              D87 D91 D96 F5 F15 F20 F25 F35 F40 F48 F52 F55 F60 F70
              F75.
       PROBLEM STRINGS AND SUBSTRINGS ARE SEQUENCES OF WORDS, READ FROM LEFT
Q8     TO RIGHT, WITH EACH WORD DIRECTLY TO THE LEFT OF THE WORD FOLLOWING
       IT.
    USED-FOR ALL.
    IS-FOR    S13 S15 S65 S70 11 12 13 14 15 16 18 19 19E 110 111 112 113
              114 115 116 117 118 119 120 121 123 125 126 1260 129 130
              D87 P20 P23 P26 M10 M20 M30 M50 M55 M60 M62 M65 M75 C2 C5
              C8 C15 C17 C20 C50 C52 C60 P2 V30 V31 V32 V33 V33R V34 V35
              V36R V37 F5 F15 F20 F25 F35 F40 F45 F48 F50 F52 F55 F60
              F70 F75 A1 A2 A3 A11 A12 A15 A17 A18 A19 A20 A24 A26 A28
              A31 A32 A34 A35 A38 A41 A42 A43 A52 A61 A62 A67 A73 A83
              A8.
       WHEN DOING A TRANSFORMATION AT A SCAN POINTER POSITION, THE FIRST
Q9     POSITION OF THE OLD STRING SHOULD BE REPLACED BY THE FIRST POSITION
       IN THE NEW STRING, TO AVOID NECESSITY FOR FIXING SCAN POINTERS;
       THIS IS NOT POSSIBLE WHEN THE TRANSFORMATION INVOLVES REMOVING OR
       CHANGING THE POSITION OF THE FIRST POSITION OF THE OLD STRING.
    USED-FOR ALL.
    IS-FOR    11 12 12E 13 14 15 16 17 18 19 19E 110 111 112 114 115 117
              118 119 120 122 124 125 126 127 128 131 132 A18 A19 A20
              A31 A32 A34 A35.
       WHEN A TRANSFORMATION WHICH IS DONE AT A SCAN POINTER POSITION
Q10    INVOLVES REMOVING OR CHANGING THE POSITION OF THE FIRST POSITION OF
       THE OLD STRING, THE SCAN POINTERS MUST BE REARRANGED SO THAT THE
       SCAN CAN RESUME PROPERLY.
    USED-FOR ALL BUT TEST1 TEST2 TEST3 TEST13 TEST18.
    IS-FOR    113 116 121 123 1260 150 151 A11 A12 A15 A17 A24 A26 A28.
    INTER     F5 F25 F40 F45 F50 F60.
Q11   WHEN A PARTICULAR STRING IS PART OF A CONDITION OF USAGE OF SOME
```

KNOWLEDGE STATEMENT, AND WHEN OTHERS MENTION IN CONDITIONS OF THE SAME TYPE A STRING THAT CONTAINS THE FIRST STRING AS A SUBSTRING, THE PROGRAM SEGMENT THAT APPLIES THE FORMER KNOWLEDGE MUST EXPLICITLY EXCLUDE THE POSSIBILITY OF THE LONGER STRING.

USED-FOR ALL.
IS-FOR I16 I19 I20 I26 I26D D82 H10.

012 IN A STRING, ANY POSITION HAS UNIQUE NEIGHBORS ON BOTH THE LEFT AND RIGHT, AND THE WORD AT THAT POSITION IS UNIQUE; WHEN ANY OF THOSE ARE CHANGED, THE OLD RELATIONS AND VALUES SHOULD BE DELETED.

USED-FOR ALL.
IS-FOR I1 I2A I3 I4 I5 I6 I7 I8 I9 I9F I10 I11 I12 I13 I14 I15 I16 I17 I18 I19 I20 I22 I24 I25 I26 I27 I28 I31 I32 D14 D17 H20 H30 H50 H55 H60 H62 H65 F45 F48 F50 F52 F55 A11 A12 A15 A17 A18 A19 A20 A31 A32 A34 A35 A50 A61 A75 A85.

013 DELETE OLD ATTRIBUTES OF A CHUNK WHEN IT IS SPLIT AND RENAMED TO OTHER CHUNKS.

USED-FOR ALL.
IS-FOR H10 C10 C15 C17 C25 C50 C52 C55.

014 DURING A SCAN PROCESS, WHEN A CONDITION IS STATED IN TERMS OF THE POSSIBLE OUTPUT OF SOME PROCESS THAT IS APPLIED AT EACH SCAN POINT, THE TEST FOR THAT CONDITION AT A PARTICULAR POINT SHOULD BE DEFERRED UNTIL THE SCAN HAS PASSED THE POINT.

USED-FOR ALL.
IS-FOR S10 S13 S15.
INTER I25 I35 I40 I20.

015 WHEN A SEQUENCE OF ACTIONS IS TO BE PERFORMED, MORE FLEXIBILITY IN ALTERING THE COURSE OF THAT SEQUENCE RESULTS BY BREAKING IT INTO SEPARATE STEPS, EACH REQUIRING AN INITIATE SIGNAL AND HAVING A COMPLETION SIGNAL; THIS BREAKING INTO STEPS IS ESPECIALLY USEFUL FOR LONGER SEQUENCES WHERE UNDER VARIOUS CONDITIONS, DIFFERENT ELEMENTS OF THE SEQUENCE ARE ACTUALLY EXECUTED.

USED-FOR ALL.
IS-FOR S10 S13 S15 S16 S17 S18 S35 S40 I5 I20 I21.

016 WHEN SOME PARTS OF AN ACTION ARE CONDITIONAL ON VARIOUS ASPECTS WHILE OTHER PARTS ARE THE SAME, COMMON TO ALL CONDITIONS, AND WHEN THE COMMON PARTS ARE FAIRLY COMPLEX, IT'S BEST TO COLLECT THE COMMON PARTS INTO A SINGLE PROGRAM SEGMENT, WITH THE CONDITIONAL PARTS SEPARATED.

USED-FOR ALL.
IS-FOR S65 S70 H10 H20 H30 H50 H55 C2 C5 C8 C10 C15 C17 C25 C50 C52 C55 C60 C80 C85 V15 V50 V60 V65 V80 V85 V90 F15 F20 F35 F70 F75 F80 A15 A17 A53 A62 A64.

017 SPLITTING MEANS THAT ADJACENCIES AT THE SPLIT POINT NO LONGER HOLD.

USED-FOR ALL.
IS-FOR C50.

018 SPECIFIC SIGNALS, UPON USE, SHOULD BE DELETED, IF THERE CAN BE NO FURTHER USE BECAUSE USE CONSISTS OF CHANGES TO NECESSARY CONDITIONS, OR IF THE USE AMOUNTS TO RECORDING THE INTENT OF THE SIGNAL FOR USE ELSEWHERE.

USED-FOR ALL.
IS-FOR S10 S65 P30 P35 P40 P45 P50 P65 P70 P75 H20 H30 H50 H55 C20 C22 C60 C80 C85 P2 P4 P6 P7 P8 P9 V20 V21 V23 V24 V25 V30 V31 V32 V33 V33A V34 V35 V36A V37 V40 V55 V60 I20 I35 I70 A1 A2 A8 A11 A12 A15 A17 A18 A19 A20 A21 A26 A28 A31 A32 A34 A35 A38 A41 A42 A43 A52 A56 A57 A58 A63 A63I A64 A66 A67 A75 A85 B1 B3 B6 13 15 17 I11 I13 I15 I17 I21 I31 I33 I41 I43.

019 WHEN SOME PROCESS REQUIRES THE OUTPUT OF ANOTHER PROCESS, AND IT IS ALSO DESIRED THAT OTHER OUTPUTS OF THAT PROCESS ARE AVAILABLE WHEN THE FIRST PROCESS STARTS, INITIATE THE PROCESSING OF THE SECOND ON ITS INPUTS IN THE REVERSE ORDER OF THEIR USE BY THE FIRST; THIS AVOIDS HAVING TO CHECK EXPLICITLY FOR THE OTHER OUTPUTS.

USED-FOR ALL.
IS-FOR I29 I30 C80.

020 WHEN SOME ENTITY IS BROKEN UP INTO PIECES, WHICH LATER BECOME PARTS OF A CORRESPONDING TREE-STRUCTURED ENTITY, THE PARENT-DAUGHTER STRUCTURE OF THE PIECES MUST BE RECORDED, AS WELL AS THE ORDERING OF THE PIECES RELATIVE TO EACH OTHER; A DISTINCTION MUST ALSO BE MADE AS TO WHETHER THE RESULTING PIECES ARE READY TO BE FORMED INTO THE TREE STRUCTURE OR WHETHER FURTHER PROCESSING IS TO BE DONE BEFORE THAT.

USED-FOR ALL.
IS-FOR S10 C25 C15 C50 C60 C70 C75 C78 C80.

021 WHEN A NEW LINKED STRUCTURE IS BEING FORMED BY REMOVING A PART OF ANOTHER STRUCTURE, AND WHEN ONE BOUNDARY OF THE NEW STRUCTURE IS NOT A BOUNDARY OF THE OLD STRUCTURE, BUT THE OTHER NEW BOUNDARY IS A BOUNDARY OF THE OLD, THE NEW BOUNDARY SHOULD BE NOTED, AND THE REST OF THE NEW STRUCTURE CAN THEN BE FORMED BY A COPYING PROCESS WHICH TERMINATES AT THE OTHER BOUNDARY.

USED-FOR ALL.
IS-FOR C60 P2 P4 P6 P7 P8 P9.

022 WHEN A CONDITION OR ACTION IS STATED IN TERMS OF TWO SEGMENTS OF A STRING BEING SCANNED, AND WHEN THE TWO SEGMENTS ARE SEPARATED BY SOME STRING OF UNKNOWN ARBITRARY LENGTH, WHEN THE EARLIER SEGMENT IS RECOGNIZED A DATA SIGNAL SHOULD NOTE THIS, AND ACTION ON THE LATER SEGMENT DEPENDENT ON IT SHOULD BE DEFERRED UNTIL THE SCAN REACHES THE LATER SEGMENT.

USED-FOR ALL.
IS-FOR I21 I22 I23 I24 I26 I26D I29 I31 I32 F5 F15 I20 F25 I35 F40 F45 F48 I50 F52 F55 I60 F70 F75.

023 THE LEFT AND RIGHT BOUNDARIES OF CHUNKS SHOULD BE MARKED RATHER THAN COMPUTED WHEN NEEDED; WHEN SOME TRANSFORMATION REMOVES PARTS OF A CHUNK, A CHECK NEEDS TO BE MADE ON WHETHER THE BOUNDARIES HAVE BEEN CHANGED, AND IF SO, IT SHOULD BE NOTED BY REMOVING THE OLD AND ADDING THE NEW.

USED-FOR ALL.
IS-FOR S10 S40 S65 I50 M20 M30 M50 M55 C15 C17 C22 C50 C52 C60 P4 P8 I5 F15 I20 F25 F35 I40 F45 I50 I60 F70 F75.
INTER I51.

024 WHEN THERE ARE MANY MORE WAYS OF COMPLETING A PROCESS EVOKED BY AN INITIATE SIGNAL THAN WAYS OF INITIATING IT, THE COMPLETION SIGNAL SHOULD BE EMITTED AT THE SAME TIME AS THE INITIATE SIGNAL, IN SUCH A WAY THAT THE INITIATE SIGNAL IS EXAMINED FIRST.

USED-FOR ALL.
IS-FOR S10 S13 S15 S16 S17 S18 S35 S40.

025 WHEN A STRING SERVES AS A SEPARATOR FOR TWO OTHERS, THIS MEANS THAT ONE STRING ENDS DIRECTLY TO THE LEFT OF THE SEPARATOR, AND ANOTHER STARTS DIRECTLY TO THE RIGHT OF THE SEPARATOR.

USED-FOR TEST2 TEST5 TEST9 TEST16 TEST17 TEST19 TEST20 TEST21 TEST22 TEST23 TEST24.
IS-FOR F15 F75.

026 WHEN THE NUMBER OF WORDS SCANNED IS BEING COUNTED, AND WHEN A TRANSFORMATION REMOVES WORDS ALREADY COUNTED, THE COUNT MUST BE ADJUSTED.

USED-FOR TEST6.
IS-FOR I52 A15 A17.

027 AT THE START OF A PROCESS THAT IS TO DETERMINE THE MAXIMUM OF SOME VALUE, THE RECORDER OF THE MAXIMUM SHOULD BE INITIALIZED TO A LOW VALUE.

USED-FOR ALL.
IS-FOR P10 P15.

028 WHEN SOME PROCESS IS DONE ON ELEMENTS OF A SET IN ORDER, BUT ELEMENTS OF THE SET MAY BE ONLY PARTIALLY COMPLETE WHEN OTHERS ARE COMPLETE AND AVAILABLE, AND IF THE PROCESS IS NOT TO START UNTIL AT LEAST THE FIRST TWO ELEMENTS ARE AVAILABLE, IT IS NECESSARY TO SIGNAL AT THE BEGINNING OF THE CREATION OF THE ELEMENTS THAT THEY ARE AVAILABLE BUT INCOMPLETE AND TO USE THOSE SIGNALS IN CHECKING FOR WHETHER TO GO AHEAD; ONE CONVENIENT SIGNAL OF THIS TYPE MAY BE THE ATTRIBUTE OF THE ELEMENTS THAT DETERMINES THE ORDER.

USED-FOR ALL.
IS-FOR P10 P15.

029 WHEN A SET OF ELEMENTS IS TO BE ORDERED, ASSIGN NUMBERS TO THE ELEMENTS IN THE APPROPRIATE ORDER, FOR EASY COMPARISON.

USED-FOR ALL.
IS-FOR P1 P2 P3 P4 P5 P8 P9 M20 M30 M50 M55.

030 IN A PROCESS OF FINDING A MAXIMUM, WHEN A NEW VALUE IS FOUND, THE OLD SHOULD BE DELETED.

USED-FOR ALL.
IS-FOR S20 P20 P22.

031 WHEN A TRANSFORMATION AT A SCAN POINT BRINGS ABOUT CHANGES WHICH AFFECT PROCESSING AT THE PRECEDING POSITION, IT IS NECESSARY TO ANTICIPATE AND APPLY THE TRANSFORMATION WHEN THE SCAN IS AT THE PRECEDING POINT.

USED-FOR TEST6 TEST9.
IS-FOR A20.

032 WHEN SOME PROCESS HAS USED NUMERICAL VALUES IN PRODUCING AN ORDERING, BUT THOSE VALUES ARE ARBITRARY, CONVERT THE OUTPUT OF THE PROCESS TO SYMBOLIC SIGNALS FOR EASIER USE ELSEWHERE.

USED-FOR ALL.
IS-FOR P30 P35 P40 P45 P50 P65 P70.

033 WHEN AN ATTRIBUTE OF SOMETHING CHANGES TO A NON-COMPATIBLE ATTRIBUTE, THE OLD ONE SHOULD BE DELETED.

USED-FOR ALL.
IS-FOR S60 C70 C75 C78.

034 WHEN A DATA VALUE IS SPECIFIED AS THE "CURRENT" OR "MOST RECENT" OBJECT WITH SOME ATTRIBUTE, AND WHEN A NEW VALUE IS COMPUTED, THE OLD ONE MUST BE DELETED.

USED-FOR ALL.
IS-FOR S40 S65 C80 F15 F75 I7.

035 WHEN A DUMMY IS USED TO REPRESENT ONE EXPRESSION IN ANOTHER, AND IF AFTER SOME PROCESSING, THE VALUE OF THE FIRST IS TO BE REINSERTED INTO THE SECOND, THERE MUST BE SOME RECORD OF THE RELATION BETWEEN THE TWO FOR USE AT THAT TIME.

USED-FOR TEST4.
IS-FOR C15 C20.

036 WHEN SOME OPERATION IS TO BE DONE ON ELEMENTS OF A LINKED STRUCTURE OF

ARBITRARY LENGTH, OR ON A SET OF ARBITRARY SIZE, IT NEEDS TO BE IMPLEMENTED AS A LOOPING PROCESS THAT GOES FROM ONE END OF THE STRUCTURE TO THE OTHER, OR THAT EXHAUSTIVELY PROCESSES THE ELEMENTS OF THE SET.
USED-FOR ALL.
ES-FOR H20 H30 H50 H55 C15 C20 C50 C60 P2 P4 P6 P7 P8 P9 V15 V30 V31 V32 V33 V33R V34 A50 A51 A52 A63 A71 A81 U6.

037 WHEN A PROCESS IS TO BE TERMINATED, DELETE ITS CONTROL SIGNALS.
USED-FOR ALL.
ES-FOR V50 A68 A69.

038 A SIGNAL THAT LEADS TO DELETING UNNECESSARY SIGNALS SHOULD BE INSERTED BEFORE OTHERS.
USED-FOR ALL.
ES-FOR V15 V50 V80 A64.

039 A SIGNAL THAT MAY NOT BE USED BECAUSE OF LACK OF ENTITIES THAT MAY APPEAR LATER, BUT THAT WOULD INTERFERE OR BE REDUNDANT AT THE LATER TIME, SHOULD BE EXPLICITLY DELETED.
USED-FOR ALL.
ES-FOR [illegible].

040 WHEN A SET OF OBJECTS IS TO BE COUNTED, AND WHEN MORE THAN ONE IS AVAILABLE AT THE SAME TIME, MULTIPLE FIRINGS MUST BE EXCLUDED, TO GET THE CORRECT COUNT: THIS MAY BE DONE BY COUNTING THEM IN ORDER AND NOTING THAT EACH HAS BEEN COUNTED.
USED-FOR ALL.
ES-FOR V55.

041 IF A PROCESS HAS POSSIBLY EMITTED SIGNALS THAT ARE NOT EXAMINED AND DELETED, WHEN THE PROCESS IS FINISHED, DO A CLEANUP OPERATION ON THOSE SIGNALS.
USED-FOR ALL.
ES-FOR V60 V50 V80 V86 V90 A64 A66 A68 A69.

042 IF A TRANSFORMATION OUTPUTS SOMETHING THAT IS INPUT TO OTHERS, AND IF THE OTHERS ARE NORMALLY NOT HANDLED AFTER THE PRESENT TYPE OF TRANSFORMATION, IT IS NECESSARY TO DO SO EXPLICITLY.
USED-FOR TEST6 TEST9 TEST10.
ES-FOR A10 A19 A41.

043 WHEN A VARIABLE STRING IS BEING COLLECTED, TEST FOR ITS TERMINATION CAN BE BY THE LENGTH OF THE COLLECTED STRING, IF THE STRINGS COLLECTED ARE APPROXIMATELY UNIFORM IN LENGTH.
USED-FOR TEST6 TEST9 TEST10.
ES-FOR A41 A42 A43.

044 WHEN SCAN POINTERS MUST BE REARRANGED, AND WHEN THE NEW LOCATION OF THE SCAN POINTERS IS NOT AVAILABLE UNTIL AFTER SOME COMPUTATION, SIGNAL SO THAT THEY CAN BE REARRANGED AT THE LATER TIME, AND DO THE REQUIRED COMPUTATION BEFORE THE SIGNAL IS EXAMINED.
USED-FOR TEST9 TEST10.
ES-FOR A50 A75 A77 A85.

045 A LOOPING PROCESS EXECUTES A PROGRAM SEGMENT REPEATEDLY, RETAINING CONTROL UNTIL A TERMINATION CONDITION IS TRUE.
USED-FOR ALL.
ES-FOR P2 P4 P6 P7 P8 P9 V30 V31 V32 V33 V33R V34 A52 A53 A59 A67 A72 A75 A83 A85 B9.

046 WHEN A LIST VALUE IS BEING FORMED BY COLLECTING PIECES, WHEN A NEW VALUE IS FORMED, THE OLD SHOULD BE DELETED.
USED-FOR ALL.
ES-FOR A56 A57 A83 B2 B8.

047 WHEN THE OUTPUT OF A PROCESS IS BEING COLLECTED AS A LIST, THE FIRST TIME A VALUE IS COLLECTED REQUIRES AN INITIALIZATION OF THE LIST THAT IS USED AS THE COLLECTED RESULT.
USED-FOR TEST10.
ES-FOR A52.

048 WHEN AN ENTITY IS SCANNED, COPIES OF WHICH MAY BE USED AT LATER POSITIONS IN THE SCAN, ITS POSITION SHOULD BE NOTED AS SCANNED: IF ITS ORDER OF APPEARANCE IS ALSO USED LATER, THAT SHOULD ALSO BE RECORDED AS SCANNED.
USED-FOR TEST6 TEST9 TEST10.
ES-FOR A62.

049 WHEN A LOOPING PROCESS IS TESTING FOR A CONDITION WHICH WILL BE TRUE IF THE LOOP TERMINATES AFTER EXHAUSTING ITS RANGE, AND WHEN SOME ACTION IS TO BE TAKEN IF THE LOOP TERMINATES PREMATURELY, THE SIGNAL FOR THE LOOP SHOULD BE FOLLOWED BY A SIGNAL FOR THE PREMATURE TERMINATION ACTION: IF THE LOOP FULFILLS ITS RANGE, THE LATTER SIGNAL MUST BE REMOVED.
USED-FOR TEST6 TEST9 TEST10.
ES-FOR A63 A68 A69.

050 WHEN A NEW ENTITY IS TO BE COMPARED TO PREVIOUS ONES OF ITS TYPE, WHEN NO OTHERS OF ITS TYPE EXIST YET, AND WHEN SOME PROCESS IS TO BE DONE ON FAILURE OF THE COMPARISONS, THE SIGNAL TO INITIATE THAT PROCESS SHOULD BE EMITTED IN PLACE OF THE COMPARISON TEST INITIATION SIGNAL.
USED-FOR TEST6 TEST9 TEST10.
ES-FOR A63E

051 WHEN A SIGNAL IS TO BE EMITTED TO CLEAN UP EXTRA OCCURRENCES OF SOME SIGNAL, TO AVOID THE NECESSITY OF AN ADDITIONAL CONDITIONAL PROGRAM SEGMENT TO REMOVE THE CLEANUP SIGNAL IN CASE NONE OF THE OTHERS EXIST, MAKE SURE THERE IS AT LEAST ONE OF THE OTHERS: EMIT A DUMMY ONE IF NECESSARY: THE CLEANUP SIGNAL MUST BE REMOVED IF IT WILL UNDESIRABLY CLEAN UP FUTURE OCCURRENCES OF THE OTHER SIGNALS.
USED-FOR TEST6 TEST9 TEST10.
ES-FOR A63I.

052 WHEN COMPARING TWO STRINGS, ONE OF WHICH HAS NOT YET HAD TRANSFORMATIONS APPLIED TO IT, IF SOME SEGMENT IN ONE STRING IS THE OUTPUT OF SOME TRANSFORMATION, A CHECK MUST BE MADE AS TO WHETHER THE OTHER STRING MIGHT BE SO TRANSFORMED.
USED-FOR TEST9 TEST10.
ES-TOR A60.

053 RENAMING ONE STRUCTURE TO ANOTHER CONSISTS OF REMOVING ELEMENTS FROM THE FIRST AND ADDING THEM TO THE SECOND.
USED-FOR ALL.
ES-FOR C20 P2 P4 P6 P7 P8 P9.

>>> Z <<<.

Z1 THE FIRST INSERTION IN THE RIGHT-HAND-SIDE GOES AT THE TOP OF :SMPX.
USED-FOR ALL.
ES-FOR T1 T2 T3 T4 T5 T6 T8 T9 T9E T10 T11 T12 T13 T15 T17 T18 T19 T20 T25 T52 O1 O3 O5 O7 O9 O11 O13 O14 O15 O17 O18 O19 O91 O96 P20 P23 P26 P27 P28 P29 H10 H20 H30 H50 M55 M60 M62 M65 C17 C52 P2 P4 P6 P7 P8 P9 V15 V30 V31 V32 V33 V33R V34 A20 A31 A32 A34 A35 A52 A59 A64 A67 A73 A83.

Z2 THE FIRST TWO RIGHT-HAND-SIDE INSERTIONS ARE ORDERED AT THE TOP OF :SMPX: WHEN IT IS DESIRED TO DO ONE THING FOLLOWED BY ANOTHER, ORDER THE "INITIATE" SIGNALS ACCORDINGLY.
USED-FOR ALL.
ES-FOR S10 S13 S15 S16 S17 S18 S35 S60 S65 T16 T21 T23 T26D T29 T30 T50 T51 P10 C2 C5 C8 V50 E20 E35 E70 A11 A12 A15 A17 A18 A24 A26 A20 A41 A51 A61 A63 A75 A85.

Z3 THE FIRST THREE RIGHT-HAND-SIDE INSERTIONS ARE ORDERED AT THE TOP OF :SMPX: WHEN IT IS DESIRED TO DO ONE THING FOLLOWED BY A SECOND FOLLOWED BY A THIRD, ORDER THE "INITIATE" SIGNALS ACCORDINGLY.
USED-FOR ALL.
ES-FOR S40 C15 C50 C60 V60 T15 E75 O19.

Z4 WHEN IT IS DESIRED TO INITIATE SOME PROCESS ON ELEMENTS OF A SET OF INPUTS IN A PARTICULAR ORDER, WHEN THE INITIATION SIGNAL IS DISTINCT FOR EACH, AND WHEN POSSIBLY MORE THAN ONE OF THOSE INITIATE SIGNALS IS AVAILABLE AT THE TIME OF THE CHECK, IT IS NECESSARY TO DETERMINE THE FIRST AND SECOND ELEMENTS OF THE SET, INITIATE THE PROCESS ON THE FIRST, AND REASSERT THE ELEMENT THAT GIVES RISE TO THE CHECK FOR INITIATION ON THE SECOND SO THAT IT WILL BE EXAMINED AGAIN LATER: THERE MUST ALSO BE A SECOND PRODUCTION THAT FIRES IN CASE NO SECOND INITIATING ELEMENT EXISTS.
USED-FOR ALL.
ES-FOR P10 P15.

Z5 WHEN A PREDICATE, WHICH IS NOT A NONFLUENT, IS USED IN A PRODUCTION WITH ANOTHER PREDICATE NOT A NONFLUENT, WHEN THE ARGUMENTS OF THE TWO PREDICATES ARE INDEPENDENT, AND WHEN THE SECOND PREDICATE IS TRUE OF SOME INSTANCES THAT ARE NOT "NEW" , THERE MUST BE IN THE CONDITION OF THE PRODUCTION SOME WAY OF EXCLUDING THE INSTANCES THAT ARE NOT "NEW".
USED-FOR ALL.
ES-FOR V5 V50 E80.

Z6 WHEN A PRODUCTION INSERTS A NEW INSTANCE OF SOMETHING NOT A NONFLUENT OCCURRING IN ITS CONDITION, AND WHEN IT DOESN'T CHANGE THE CONDITION SO THAT IT WON'T MATCH, IT MUST PUT SOMETHING THAT WILL ULTIMATELY CHANGE THE CONDITION BEFORE THE REPETITION IN THE RHS.
USED-FOR ALL.
ES-FOR V5.

Z7 WHEN ELEMENTS OF A SET ARE TO BE PROCESSED IN A PARTICULAR ORDER, AND WHEN THE CHECK FOR INITIATING THE PROCESS ON AN ELEMENT IS THE SAME FOR THE ENTIRE SET, DETERMINE THE "LEAST" ELEMENT AS YET UNPROCESSED, INITIATE THE PROCESS ON IT, AND RE-ASSERT THE CHECK SIGNAL AFTER IT FOR A LATER RE-EXAMINATION.
USED-FOR ALL.
ES-FOR A5E B5.

Z8 WHEN A PRODUCTION TESTS A CONDITION AS A RESPONSE TO SOME SIGNAL, AND CHANGES CONDITIONS IN A WAY THAT OTHER CONDITIONS RELATED TO THAT SIGNAL MIGHT BECOME TRUE, IT IS NECESSARY TO ASSERT THE SIGNAL AGAIN.
USED-FOR TEST6 TEST9 TEST10.
ES-FOR A61.

Z9 WHEN A PRODUCTION DELETES VALUES AND INSERTS VALUES FOR THE SAME PREDICATE, AND WHEN SOME OF THE DELETIONS MAY BE ON VALUES THE SAME AS THE INSERTIONS, THE DELETIONS MUST BE DONE FIRST.
USED-FOR ALL.
ES-TOR T5 T3E T41.

Z10 IF A LOOP BODY CONSISTS OF THE FIRING OF ONLY ONE PRODUCTION, EACH SUCH PRODUCTION SHOULD INCLUDE THE TERMINATION CONDITION: SEPARATE

```
          PRODUCTIONS ARE NECESSARY IF THE BODY IS TO BE EXECUTED WHEN THE
          TERMINATION CONDITION IS TRUE, OR IF SOMETHING SPECIFIC IS TO BE
          DONE ON TERMINATION.
       USED-FOR ALL.
       KS-FOR    P2 P4 P6 P7 P8 P9 V30 V31 V32 V33 V33P V34 A52 A53 A59 A67
                    A73 A75 A83 A85 B8.
211    A PROCESS CAN RETAIN CONTROL BY PUTTING ITS CONTROL SIGNALS FIRST IN
          THE BUS.
       USED-FOR ALL.
       KS-FOR    P2 P6 P7 V30 V31 V32 V33 V33P V34 A52 A67 A73 A83.

218 KNOWLEDGE STATEMENTS (154 53 11)
AVERAGE USES PER PRODUCTION, N = 2.80, Q = 2.86, Z = 0.650
        WITH VARIANCES 2.75, 3.57, 0.617 (1.66, 1.89, 0.78 SQUARTD)

DISTRIBUTION OF KS'S OVER PRODUCTIONS
NUMBER OF KS'S:  0   1   2   3   4   5   6   7   8   9  10  11  12
NO. OF P'S    N  5  57  55  56  45  21   9   7   1   1   0   0   0
  BY          Q  7  65  61  29  60  11   9   4   8   1   1   0   0
  CLASS       Z 122 106  11  13   0   0   0   0   0   0   0   0   0

257 PRODUCTIONS, 27 TESTS, PROCESS TIME 11 MIN, 33.6 SEC
```