INDUCE 1.2
A Program for Learning Structural Descriptions from Examples (Program Listing)

R. Stepp

Intelligent Systems Group
Department of Computer Science
University of Illinois
Urbana, Illinois 61801-2987

March 1982

This work was supported in part by the National Science Foundation under grant NSF MCS 79-06614.

PREFACE

This document presents a listing of the program INDUCE 1.2 with cross-referencing of variables and procedures. This cross-referencing has been obtained through the use of the Pascal STYLIST program. The first version of the program INDUCE 1 was developed by J. Larson and the current version 1.2 by T. Dietterich and M. Tuceryan. The listing of the program was prepared by R. Stepp.

ECK#	DECKNAME	LENGTH	DECKS CALLED				
1)	IND1	983	PGRAPHA	ENTERPA	VLINTA	DUMPG	DUMPS
			DUMPARITH	INSIDE	ADDSEL	EXTND	ILINE
			PEOS	GETCHRR	INIT	NEWG	GIN
			GOUT	NEWCPXTAB	FREETAB	NEWC	FREECPX
			EXPLN	PMETAD	ENTERP	SOUT	ADDCONS
			ALLC	CALCARITH	SUBG1	PCPX	AQ
			CLEANCPX	LQST2	AQSET	ENTERD	VL1
			NEWGP	PGRAPH	TOKEN	VLINT	COSTG
			ADDTOMQ	TRIMG	COMPMS	TRIMM	ADDMETA
			PROCARITH	ADDML	COVER		
2)	PGRAPHA	5					
3)	ENTERPA	3					
4)	VLINTA	8					
5)	DUMPG	25					
6)	DUMPS	16					
7)	DUMPARITH	17					
8)	INSIDE	29					
9)	ADDSEL	67					
10)	EXTND	44					
11)	ILINE	24					
12)	PEOS	9					
13)	GETCHRR	7					
14)	INIT	99					
15)	NEWG	17					
16)	GIN	6					
17)	GOUT	6					
18)	NEWCPXTAB	12					
19)	FREETAB	10					
20)	NEWC	14					
21)	FREECPX	19					
22)	EXPLN	41	RDEX				
23)	RDEX	49					
24)	PMETAD	14					
25)	ENTERP	184	PDOM	PRINTPS	GETNUM		
26)	PDOM	18					
27)	PRINTPS	62	WTOLER	MAX			
28)	WTOLER	9					
29)	MAX	6					
30)	GETNUM	22					
31)	SOUT	10					
32)	ADDCONS	75					
33)	ALLC	51					
34)	CALCARITH	119					
35)	SUBG1	82	SUBG				
36)	SUBG	109	MATCH				
37)	MATCH	34					
38)	PCPX	36					
39)	AQ	246	TRIM				
40)	TRIM	76	COSTF				
41)	COSTF	100					
42)	CLEANCPX	20					
43)	LQST2	43	MAXCOUNT	ADDEDCOST	MAXCPXS		
44)	MAXCOUNT	18					
45)	ADDEDCOST	59					
46)	MAXCPXS	90					
47)	AQSET	134					
48)	ENTERD	22					

```
( 50) NEWGP          97
( 51) PGRAPH        132      WRITEVAL
( 52) WRITEVAL       20
( 53) TOKEN         136      FINDROW     FIXSYM
( 54) FINDROW        27
( 55) FIXSYM         21
( 56) VLINT         215      PROCESS
( 57) PROCESS       153      TESTGTOP    DUMPROC     PARSEARITH
( 58) TESTGTOP        5
( 59) DUMPROC        70
( 60) PARSEARITH     58      APUSH       OPUSH
( 61) APUSH          10
( 62) OPUSH           6
( 63) COSTG          70
( 64) ADDTOMQ        12
( 65) TRIMG          72
( 66) COMPMS         62
( 67) TRIMM          17
( 68) ADDMETA        58
( 69) PROCARITH      44      BUILDG
( 70) BUILDG         31      ADDLINK     FINDFUNC
( 71) ADDLINK        14
( 72) FINDFUNC       31
( 73) ADDML          59
( 74) COVER         176      ABSOURB     TOOSMALL
( 75) ABSOURB        21
( 76) TOOSMALL       37

DECK FILE HAS BEEN RESTYLED
```

```
(  1)    1  *DECK IND1.2
(  1)    2   (*  TO CONVERT FROM TEST VERSION OF PROGRAM TO PRODUCTION VERSION:
(  1)    3       USE THE BOSS COMMAND:  VS/$D+/DEBUGOFF/W
(  1)    4       TO CONVERT FROM PRODUCTION VERSION OF PROGRAM TO TEST VERSION:
(  1)    5       USE THE BOSS COMMAND:  VS/DEBUGOFF/$D+/W *)
(  1)    6   (*DEBUGOFF     TURN ON PASCAL DEBUGGING SYSTEM
(  1)    7
(  1)    8     ALSO NOTE THAT COLUMN ONE OF THE INPUT FILE IS RESERVED FOR CARRIAGE
(  1)    9     CONTROL AND CANNOT BE USED FOR ANYTHING ELSE.  ALL CARRIAGE CONTROL
(  1)   10     IS PLACED IN COMMENT BRACKETS SO THE COMPILER WILL IGNORE IT.
(  1)   11  1
(  1)   12                  I N D U C E - 1  --  VL21 RULE INDUCTION
(  1)   13                       VERSION AS OF OCT 13, 1978
(  1)   14   *)
(  1)   15
(  1)   16    (*
(  1)   17             AUTHOR:           JAMES B. LARSON
(  1)   18             MODIFIED BY:      THOMAS G. DIETTERICH (SEE MODIFICATION HISTORY BELOW)
(  1)   19             DATE-WRITTEN:     JUNE, 1977
(  1)   20             SOURCE COMPUTER:  CONTROL DATA CYBER 175
(  1)   21             COMPILER:         PASCAL CYBER V2.0
(  1)   22                               UNIVERSITY OF MINNESOTA  (76/04/02)
(  1)   23             OBJECT COMPUTER:  CONTROL DATA CYBER 175
(  1)   24             INSTALLATION:     UNIVERSITY OF ILLINOIS,  CSO
(  1)   25             REMARKS:
(  1)   26                SUPPORTED IN PART BY GRANTS FROM THE NATIONAL SCIENCE FOUNDATION.
(  1)   27
(  1)   28             INTRODUCTION:
(  1)   29
(  1)   30
(  1)   31        THIS PROGRAM SYNTHESIZES VL21 FORMULAS (REPRESENTED AS DECISION RULES)
(  1)   32        WHICH ARE GENERALIZATIONS OF A SET OF OF VL21 FORMULAS.  ASSUMPTIONS
(  1)   33        ARE THE FOLLOWING:
(  1)   34
(  1)   35        1. ALL VARIABLES ARE EXISTENTIALLY QUANTIFIED AND REPRESENT
(  1)   36        DISTINCT VALUES OF THEIR DOMAIN.
(  1)   37
(  1)   38        2. EACH EXPRESSION IS ASSUMED TO BE A PRODUCT OF SELECTORS IN VL21
(  1)   39        WITH ATOMIC FORMS WHICH ARE FUNCTIONS OF SIMPLE VARIABLES
(  1)   40
(  1)   41        3. EXPRESSIONS ARE REQUIRED TO BE IN A FORM WHICH CAN BE TRANSLATED
(  1)   42        INTO A CONNECTED GRAPH.  MORE PREDICATES MAY BE ADDED BY THE USER TO ASSURE
(  1)   43        THIS.
(  1)   44
(  1)   45             THE PROGRAM GENERATES LARGER AND LARGER PRODUCTS OF SELECTORS
(  1)   46        WHICH COVER A SPECIFIC ELEMENT OF THE SET OF FORMULAS WHICH ARE
(  1)   47        TO BE COVERED.  WHEN ONE PRODUCT IS FOUND WHICH DOESN'T COVER
(  1)   48        ANY FORMULA IN OTHER SETS, AN AQVAL/1 TYPE PROCEDURE IS CALLED
(  1)   49        TO EXTEND THE REFERENCES.  COVERING IS TESTED BY A SUBGRAPH
(  1)   50        MATCHING ALGORITHM WHICH FINDS A SPANNING TREE OF THE SMALLER
(  1)   51        OF THE TWO GRAPHS AND TRIES TO FIND A TREE IN THE LARGER
(  1)   52        GRAPH WHICH MATCHES.  A BACKTRACK MECHANISM IS BUILT IN TO
(  1)   53        TO BACK DOWN THE TREE IF SOME MATCH FAILS.
(  1)   54
(  1)   55
(  1)   56           ANY DESCRIPTOR FOLLOWED BY A NUMBER IS A DUMMY VARIABLE.
(  1)   57
(  1)   58
```

```
(  1)   59
(  1)   60   USING VL21 ON THE CYBER
(  1)   61
(  1)   62   FILE USAGE:
(  1)   63       THERE ARE SEVERAL FILES WHICH THE PROGRAM USES.  THEY ARE BRIEFLY DESCRIBED
(  1)   64   BELOW:
(  1)   65
(  1)   66   IFILE - INPUT FROM TTY
(  1)   67   OUTPUT - OUTPUT TO TTY
(  1)   68   STAB - SYMBOL TABLE - TO START ON A NEW PROBLEM, THIS FILE MAY BE EMPTY.
(  1)   69       THE PROGRAM LOADS DESCRIPTORS INTO THIS FILE AT THE END OF EACH
(  1)   70       SESSION (Q-COMMAND).
(  1)   71   TABLES - PARSE TABLE WHICH CONTAINS VALID SYNTAX OF VL21 EXPRESSIONS
(  1)   72   GFILE - STORAGE OF INTERNAL RULE FORMAT.  THE Q-COMMAND AUTOMATICALLY
(  1)   73       STORES RULES INTO GFILE AND THE SYMBOL TABLE INTO STAB.  WHEN THE
(  1)   74       PROGRAM IS RERUN, THE RULES AND STAB ARE READ BACK INTO CORE
(  1)   75   CFILE - OPTIONAL COMMAND FILE.  IF COMMANDS ARE HERE, THE FIRST LINE MUST
(  1)   76       BE BLANK FOLLOWED BY LINES OF INPUT A ONE WOULD ENTER ON THE TERMINAL
(  1)   77
(  1)   78
(  1)   79   COMPILING THE PROGRAM:
(  1)   80
(  1)   81       BUILD A PROCEDURE FILE WHICH PERFORMS THE FOLLOWING COMMANDS:
(  1)   82
(  1)   83         $GET,VL21.                SOURCE FOR PROGRAM
(  1)   84         $GET,PASINI/UN=LIBRARY.   PASCAL INITIALIZATION PROCEDURE
(  1)   85         $CALL,PASINI.             INITIALIZE FOR PASCAL.
(  1)   86         $SETTL,10.                SET TIME LIMIT
(  1)   87         $PASCAL,VL21,L,V.         L=LISTING, V=OBJECT MODULE.
(  1)   88         $LOAD(V).                 INVOKE THE LOADER.
(  1)   89         $LDSET(LIB=PASCLIB).      ALLOW REFERENCES TO THE PASCAL LIBRARY
(  1)   90         $NOGO(LVL21).             CREATE LOAD MODULE ON FILE LVL21.
(  1)   91         $SAVE,LVL21.              SAVE THE LOAD MODULE AS A PERMANENT FILE.
(  1)   92
(  1)   93
(  1)   94
(  1)   95   RUNNING THE PROGRAM:
(  1)   96
(  1)   97       BUILD A PROCEDURE FILE WHICH PERFORMS THE FOLLOWING COMMANDS:
(  1)   98         $GET,LVL21.               LOAD MODULE FOR VL21  (SEE ABOVE)
(  1)   99         $GET,TABLES.
(  1)  100         $GET,EXPLAIN.             INPUT FILES FOR THE PROGRAM.
(  1)  101         $GET,CFILE.               (IF CFILE IS TO BE USED TO INPUT EVENTS)
(  1)  102         $RFL,120000.              RFL IS VARIABLE DEPENDING ON THE SIZE OF
(  1)  103                                   THE PROBLEM (SEE LIMITATIONS SECTION BELOW).
(  1)  104         $REDUCE(-)                SET "NO REDUCE" MODE FOR PASCAL PROGRAMS.
(  1)  105         $ASSIGN,TT=IFILE.         ASSIGN LFN "IFILE" TO THE TERMINAL.
(  1)  106         LVL21.                    EXECUTE THE PROGRAM.
(  1)  107
(  1)  108
(  1)  109
(  1)  110   BUILDING THE RULE BASE:
(  1)  111       SCRATCH STAB AND GFILE (RETURN,STAB,GFILE).
(  1)  112       BUILD A COMMAND FILE (CFILE).
(  1)  113            USE COMMANDS M AND A AS FOLLOWS:
(  1)  114                 <BLANK>
(  1)  115                 M       <COMMAND TO MODIFY RULE BASE>
(  1)  116                 A       <COMMAND TO ADD RULE TO BASE>
```

```
(  1)  117                [SH(X1)=1][SH(X2)=1][P(X1,X2)=1]     <PART OF RULE>
(  1)  118                =>[D=1].                             <CONSEQUENCE>
(  1)  119                M
(  1)  120                A
(  1)  121                [SH(X1)=2][SH(X2)=1][P(X1,X2)=1]=>[D=2].
(  1)  122
(  1)  123
(  1)  124           ALWAYS TERMINATE A RULE WITH A PERIOD.  PREMISE SHOULD FORM A
(  1)  125           CONNECTED C-GRAPH (CONJUNCTIVE GRAPH) WHEN TRANSLATED, CONSEQ
(  1)  126           SHOULD BE A SELECTOR WITHOUT ARGUMENTS.
(  1)  127
(  1)  128
(  1)  129
(  1)  130    EXAMINING OR DELETING RULES:
(  1)  131       AFTER BUILDING THE RULE BASE, RUN THE PROGRAM AND ENTER
(  1)  132       THE COMMAND M FOLLOWED WITH D.  IN RESPONSE TO
(  1)  133       THE QUESTION "DELETE RULE", ENTER Y (DELETE THE RULE JUST PRINTED
(  1)  134       OUT), N (DON'T DELETE THIS RULE) OR Q (RETURN TO COMMAND LEVEL).
(  1)  135
(  1)  136
(  1)  137
(  1)  138    CHANGE PARAMETERS:
(  1)  139          ENTER THE P COMMAND AND THEN THE PARAMETERS WHEN ASKED.
(  1)  140
(  1)  141
(  1)  142    ADD DOMAIN STRUCTURES:
(  1)  143           ENTER E COMMAND AND THEN THE STRUCTURE.  THESE STRUCTURES ARE
(  1)  144       NOT CURRENTLY STORED FROM ONE EXECUTION TO THE NEXT.  ENTER
(  1)  145       THE STRUCTURE AS FOLLOWS:
(  1)  146
(  1)  147                E   <E COMMAND>
(  1)  148                [SH=1,2,3,5]=>[SH=10].
(  1)  149
(  1)  150
(  1)  151           NOTE THAT THE DESCRIPTORS ARE GIVEN WITHOUT ARGUMENTS AND THAT
(  1)  152       ELEMENTS IN THE REFERENCE ARE SEPARATED BY COMMAS.  THE ENTIRE
(  1)  153       RULE IS TERMINATED WITH A PERIOD.
(  1)  154
(  1)  155
(  1)  156
(  1)  157    COVER SET OF RULES:
(  1)  158           ENTER THE C COMMAND AND THEN THE SET WHICH IS TO BE COVERED.
(  1)  159
(  1)  160       THE PROGRAM PRINTS OUT INTERMEDIATE RESULTS:
(  1)  161       1. EACH CONSISTENT FORMULA IS PRINTED AS IT IS FOUND
(  1)  162       2. IF IT IS NOT ALREADY IN THE STAR, THEN THE GENERALIZATION
(  1)  163       OF THE FORMULA IS PRINTED ALONG WITH STEPS IN THE GENERALIZATION
(  1)  164       PROCESS
(  1)  165       3. THE RULE WHICH IS SELECTED IS PRINTED AND ALL FORMULAS
(  1)  166       WHICH ARE COVERED BY THIS FORMULA ARE LISTED.
(  1)  167
(  1)  168
(  1)  169   LIMITATIONS AND CONSTRAINTS:
(  1)  170       THE PROGRAM HAS CERTAIN LIMITATIONS INTRODUCED BY THE CONSTANTS AVAILABLE
(  1)  171       IN THE "CONST" SECTION OF THE MAIN PROGRAM.  THESE CONSTANTSCONTROL
(  1)  172       THE SIZES OF MANY STRUCTURES WHICH MAY BE SENSITIVE TO THE CURRENT
(  1)  173       PROBLEM CHARACTERISTICS.  THESE CONSTANTS MAYBE INCREASED (TO ALLOW
(  1)  174       LARGER DATA STRUCTURES) OR DECREASED (TO PERMIT MORE COPIES OF A DATA
```

```
(  1)  175       STRUCTURE IN MEMORY AT ONE TIME).  THE CONSTANTS AND THEIR USE APPEAR
(  1)  176       BELOW (SUGGESTED VALUES ARE IN PARENTHESES):
(  1)  177
(  1)  178       SYMSZE(36) - IS THE SIZE OF THE SYMBOL TABLE.  IT CAN BE ESTIMATED BY
(  1)  179       FINDING THE SYM OF THE NUMBER OF FUNCTIONS, PREDICATES, AND
(  1)  180       DISTINCT VARIABLES PLUS THE NUMBER OF GROUPS OF VARIABLES PLUS 2
(  1)  181       (FOR #PT AND FORALL), PLUS 2 TIMES THE NUMBER OF BINARY PREDICATES
(  1)  182       (FOR MST-, LST- TYPE PREDICATES).  IN VL1 MODE, SYMSZE IS THE NUMBER
(  1)  183       OF VL1 VARIABLES PLUS 1.  AN EXTRA ENTRY SHOULD BE RESERVED FOR THE
(  1)  184       DECISION CLASS SYMBOL IN VL21 MODE.
(  1)  185
(  1)  186       NDES(15) - IS THE SIZE OF THE DSTRUC TABLE. (DOMAIN STRUCTURES)  ONE ROW
(  1)  187       IS REQUIRED IN THIS TABLE FOR EACH INTERNAL NODE IN EACH GENERALIZATION
(  1)  188       STRUCTURE (I.E. ONE ROW FOR EACH RULE WHICH IS INPUT WITH THE "E" COMMAND).
(  1)  189
(  1)  190       GSIZE(30) - SPECIFIES THE SIZE OF ALL GRAPH STRUCTURES IN THE PROGRAM
(  1)  191       AND THE NUMBER OF VL1 TYPE VARIABLES WHICH ARE ALLOWED IN THE PROGRAM.
(  1)  192       GSIZE IS ALSO THE SIZE OF THE METASELECTOR TABLE.  THIS NUMBER BEING
(  1)  193       TOO SMALL IS PROBABLY THE CAUSE OF AN "ARRAY INDEX OUT OF BOUNDS"
(  1)  194       MESSAGE AND MAY BE ESTIMATED BY FINDING THE SUM OF THE NUMBER OF
(  1)  195       SELECTORS IN THE LONGEST RULE WHICH MUST BE STORED, PLUS THE NUMBER
(  1)  196       OF VARIABLES IN THE RULE PLUS 1 (NOT INCLUDING META SELECTORS).  BE
(  1)  197       SURE TO INCLUDE IN THIS COUNT THE NUMBER OF METASELECTORS ADDED AS
(  1)  198       MST-, LST-, AND EQUIVALENCE (=SAME) TYPE SELECTORS OR AS THE RESULT OF
(  1)  199       RESTRICTION RULES.  AN ESTIMATE WHICH IS TOO LARGE WILL USE UP MEMORY
(  1)  200       VERY QUICKLY AND CAUSE A MESSAGE "STACK OVERRUNS HEAP".  THEREFORE,
(  1)  201       THE PARAMETER SHOULD BE APPROXIMATED RATHER CLOSELY.
(  1)  202
(  1)  203       MNVAL(15) - IS THE MAXIMUM VALUE OF A REFERENCE OF ANY SELECTOR.
(  1)  204       EACH REFERENCE IS ALLOWED TO CONTAIN AT MOST THE SET OF VALUES 0..MNVAL.
(  1)  205       THERE IS A MAXIMUM VALUE OF THIS PARAMETER DETERMINED BY THE
(  1)  206       ARCHITECTURE OF THE MACHINE (WORD SIZE).  CDC IS 58.  DEC IS ABOUT 30.
(  1)  207
(  1)  208       MLNK(18) - IS THE NUMBER OF LINKS TO ANY NODE OF A GRAPH STRUCTURE.
(  1)  209       THIS MAY BE ESTIMATED BY FINDING THE MAXIMUM NUMBER OF TIMES THAT A
(  1)  210       PARTICULAR VARIABLE OCCURS IN A RULE AND USING EITHER THIS FIGURE OR
(  1)  211       THE LARGES NUMBER OF ARGUMENTS OF ANY ONE FUNCTION, WHICHEVER IS
(  1)  212       LARGEST.  MLNK MUST BE ONE LARGER THAN EITHER OF THESE NUMBERS SINCE
(  1)  213       LINKS ARE STORED AS AN ARRAY OF NUMBERS WHICH TERMINATES IN A 0 VALUE.
(  1)  214
(  1)  215
(  1)  216       WHEN THE PROGRAM REACHES ONE OF THESE LIMITS, IT ATTEMPTS TO GIVE AN
(  1)  217       INFORMATIVE MESSAGE INDICATING WHICH LIMIT IT "RAN INTO" BEFORE IT ABORTS.
(  1)  218       TO REPAIR THIS SITUATION, INCREASE THE APPROPRIATE CONSTANT AND RECOMPILE
(  1)  219       THE PROGRAM.
(  1)  220
(  1)  221
(  1)  222
(  1)  223   MODIFICATION HISTORY:
(  1)  224
(  1)  225       5/5/78  MAJOR SET OF REVISIONS COMPLETED AND DOCUMENTATION [2] REVISED.
(  1)  226               REVISIONS INCLUDED PROGRAM FIXES TO SEVERAL ROUTINES,
(  1)  227               IMPROVED DOCUMENTATION, TERMINAL INTERACTION, NEW VL COST
(  1)  228               FUNCTION #5, NEW METASELECTOR PRINTOUTS (PGRAPH), BETTER
(  1)  229               SYNTAX ERROR DIAGNOSTICS (VLINT), AND LIMITATIONS DOCUMENTATION.
(  1)  230               TOM DIETTERICH UNIVERSITY OF ILLINOIS, DEPT OF COMPUTER SCIENCE.
(  1)  231
(  1)  232   09/01/80 - INPUT/OUTPUT REVISIONS DONE TO RULE PRINTING ROUTINE AND THE
```

```
1)  233              THE 'COVER' ROUTINE.
1)  234
1)  235
1)  236
1)  237    REFERENCES:
1)  238
1)  239         [1]  INDUCTIVE INFERENCE IN THE VARIABLE VALUED PREDICATE LOGIC SYSTEM
1)  240              VL21 : METHODOLOGY AND COMPUTER IMPLEMENTATION, JAMES LARSON, PHD
1)  241              THESIS, UNIVERSITY OF ILLINOIS REPORT UIUC DCS-R-77-869, MAY 1977.
1)  242
1)  243         [2]  INDUCE-1: AN INTERACTIVE INDUCTIVE INFERENCE PROGRAM IN VL21 LOGIC
1)  244              SYSTEM, JAMES LARSON, UNIVERSITY OF ILLINOIS REPORT UIUCDCS R-77-876
1)  245              MAY 1977.
1)  246
1)  247
1)  248
1)  249
1)  250  1
1)  251                  I N D U C E - 1  --  DEFINITIONS
1)  252   *)
1)  253  PROGRAM VL21(IFILE/,OUTPUT/,STAB,GFILE,TABLES,CFILE,EXPLAIN,VL1EVE,INPUT/);
1)  254
1)  255  LABEL
1)  256        1,
1)  257        2,
1)  258        3,
1)  259        4,
1)  260        5,
1)  261        99;
1)  262
1)  263  CONST                (* SEE LIMITATIONS DOCUMENTATION ABOVE FOR MORE INFORMATION CONCERNING THESE CONSTANTS *)
1)  264        SYMSZE          = 100;(*# OF DESCRIPTORS +# OF DUMMY VARIABLES +10 => # ROWS IN STAB*)
1)  265        NDES            = 15;(*NUMBER OF ENTRIES IN DSTRUC RECORD *)
1)  266        GSIZE           = 62;(*# OF DUMMY VBLS + # SELECTORS IN AN EVENT + 10 =>
1)  267                                 # NODES IN G  *)
1)  268        MNVAL           = 19;(* MAXIMUM VALUE IN ANY DOMAIN(0..MNVAL)*)
1)  269        MLNK            = 18;(* MAXIMUM # OF LINKS TO ANY NODE +1*)
1)  270        MRULE           = 10;(*MAX # OF RULES IN EITHER F1 OR F0 *)
1)  271        MAXPTSIZE       = 44;(* SIZE OF THE PARSE TABLE IN FILE "TABLES" *)
1)  272        MAXASTACK       = 20;(* MAX SIZE OF ARITHMETIC EXPRESSION STACKS = # OPERATORS + VALUES IN EXPRESSION *)
1)  273        VLRULE          = 1;(* PARSE TABLE PRODUCTION FOR VLRULE *)
1)  274        ARITHDD         = 22;(* PARSE TABLE PRODUCTION FOR ARITH DERIVED DESCRIPTOR *)
1)  275        MAXCSTF         = 7;(* MAXIMUM NUMBER OF COST FUNCTIONS (EITHER AQ OR VL) *)
1)  276
1)  277  TYPE
1)  278
1)  279        GPTR            = ^GRAPH;(* DEFINE POINTERS FOR THESE RECORDS *)
1)  280        DPTR            = ^DSTRUC;
1)  281        PPTR            = ^PT;
1)  282        SPTR            = ^SYMTAB;
1)  283        APTR            = ^AQPARM;
1)  284        CPTR            = ^CPX;
1)  285        CPXTABPTR       = ^CPXTABLE;
1)  286        ARITHPTR        = ^ARITHSTACK;
1)  287        ACTIONCODE      = (ADD, SUBTRACT, MULTIPLY, DIVIDE, MODULO, MINUS, FUNCT, NUMBER );
1)  288                                  (* ACTION CODES TO BE TAKEN BY ARITHMETIC INTERPRETER *)
1)  289
1)  290        ARITHSTENTRY    = RECORD          (* ARITHMETIC STACK ENTRY *)
```

```
( 1)  291          ACTION          : ACTIONCODE;
( 1)  292          ARGUMENT        : INTEGER;      (* CONTAINS PNO OF FUNCTION OR VALUE OF NUMBER *)
( 1)  293          DUMMY           : PACKED ARRAY[1..MLNK] OF 0..SYMSZE;(* PNOS OF DUMMY VARIABLES *)
( 1)  294                            (* ZERO TERMINATES THE LIST *)
( 1)  295                        END;(*ARITHSTENTRY*)
( 1)  296
( 1)  297        ARITHSTACK      = RECORD            (* ARITHMETIC STACK *)
( 1)  298          APPLIED         : BOOLEAN;      (* TRUE IF THIS HAS BEEN APPLIED TO RULE BASE *)
( 1)  299          SIZE            : 0..MAXASTACK;(* NUMBER OF ENTRIES IN USE *)
( 1)  300          NEXT            : ARITHPTR; (* WE KEEP ARITH STACK ENTRIES IN LINKED LIST *)
( 1)  301          STACK           : ARRAY[1..MAXASTACK] OF ARITHSTENTRY;(* THIS IS STACK *)
( 1)  302                        END;(* ARITHSTACK *)
( 1)  303
( 1)  304        PT              = RECORD                (* PARSE TABLE MAXPTSIZE ROWS, ONE ROW FOR
( 1)  305                                                  EACH PRODUCTION *)
( 1)  306          RHS             : ARRAY[1..MAXPTSIZE,1..13] OF INTEGER;(* RIGHT HAND SIDE OF PRODUCTION,
( 1)  307                                                  CODES ARE EITHER A CHARACTER
( 1)  308                                                  -(ANOTHER PRODUCTION),
( 1)  309                                                  OR ZERO TO END THE PRODUCTION. *)
( 1)  310          CONT            : ARRAY[1..MAXPTSIZE] OF BOOLEAN;(* CONTINUATION FLAG *)
( 1)  311          SRULE           : ARRAY[1..MAXPTSIZE] OF INTEGER;(* SEMANTIC RULE TO EXECUTE *)
( 1)  312          NAME            : ARRAY[1..MAXPTSIZE,1..60] OF CHAR(*  NAME OF THIS PRODUCTION *)
( 1)  313                        END;
( 1)  314        VALTP           = SET OF 0..MNVAL;      (* VALUE TYPE--SET OF POSSIBLE VALUE
( 1)  315                                                  THAT A FUNCTION CAN TAKE ON *)
( 1)  316        NODEA           = PACKED ARRAY[1..MLNK] OF 0..GSIZE;(*TYPE FOR NODE LIST*)
( 1)  317
( 1)  318        CPXTABLE        = RECORD              (* COMPLEX TABLE *)
( 1)  319          COVLIST         : ARRAY[1..MRULE] OF INTEGER;(* LIST OF RNO'S OF RULES FROM WHICH THIS CPX *)
( 1)  320                                                (* WAS DERIVED *)
( 1)  321          COVCOUNT        : INTEGER;            (* NUMBER OF ELEMENTS IN COVLIST *)
( 1)  322          NXT             : CPXTABPTR           (* POINTER TO NEXT CPXTABLE, USED TO MAINTAIN FREE LIST *)
( 1)  323                        END;
( 1)  324
( 1)  325        CPX             = RECORD                (* COMPLEX -- USED IN VL1 PART OF
( 1)  326                                                  PROGRAM *)
( 1)  327          COST            : INTEGER;(* COST OF COMPLEX *)
( 1)  328          FQ              : BOOLEAN;(* LIST OF COMPLEXES NOT COVERED BY ANY LQ *)
( 1)  329          FP              : BOOLEAN;(* LIST OF COMPLEXES NOT COVERED BY ANY STAR *)
( 1)  330          CVAL            : PACKED ARRAY[1..GSIZE] OF VALTP;(* SELECTOR VALUES *)
( 1)  331          TAB             : CPXTABPTR;          (* POINTER TO ASSOCIATED TABLE *)
( 1)  332          NXTC            : CPTR(* POINTER TO NEXT COMPLEX *)
( 1)  333                        END;
( 1)  334        GRAPH           = RECORD                (* GENERAL GRAPH STRUCTURE USED IN
( 1)  335                                                  VL21 PART OF PROGRAM *)
( 1)  336          COEF            : INTEGER;      (* UNUSED -- COEFFICIENT OF ENTIRE C
( 1)  337                             FORMULA *)
( 1)  338          RNO             : INTEGER;(*RULE NUMBER*)
( 1)  339          FP              : BOOLEAN;(*TEMPORARY FLAG USED IN COVER PROCEDURE*)
( 1)  340          MSEL            : CPTR;         (* POINTER TO THE METASELECTORS *)
( 1)  341          COST            : ARRAY[1..5] OF INTEGER;(*COST OF THIS FORMULA*)
( 1)  342          ESET            : VALTP;        (* EVENT SET OF THIS GRAPH *)
( 1)  343          VBL             : PACKED ARRAY[1..GSIZE] OF BOOLEAN;(* TRUE IF ENTRY IS DUMMY *)
( 1)  344          ORDIRR          : PACKED ARRAY[1..GSIZE] OF BOOLEAN;(* ORDER OF ARGS IRRELEVENT *)
( 1)  345          VAL             : PACKED ARRAY[1..GSIZE] OF VALTP;(* VALUE OF THIS NODE *)
( 1)  346          COUNT           : PACKED ARRAY[1..GSIZE] OF INTEGER;(* NO OF TIMES USED IN NEWG *)
( 1)  347          ASSGN           : PACKED ARRAY[1..GSIZE] OF 0..GSIZE;(* ASSIGNMENT OF NODE *)
( 1)  348          PNO             : PACKED ARRAY[1..GSIZE] OF -SYMSZE..SYMSZE;(* DESC NUMBER *)
```

```
(  1)  349          DPNO            : PACKED ARRAY[1..GSIZE] OF 0..SYMSZE;(* DESC NUMBER OF ACTUAL
(  1)  350                                                  DUMMY VARIABLE ENTRY *)
(  1)  351          DUMNUM          : PACKED ARRAY[1..GSIZE] OF 0..SYMSZE;(* WORK PACKED ARRAY *)
(  1)  352          NXTN            : GPTR;(* POINTER TO NEXT GRAPH *)
(  1)  353          NNEG            : GPTR;(* POINTER TO NEG GRAPH *)
(  1)  354          LNK             : ARRAY[1..GSIZE] OF NODEA(*LINKS FOR NODES*)
(  1)  355                        END;
(  1)  356       SYMTAB           = RECORD          (* SYMBOL TABLE -- INDEXED BY "PNO" ENTRIES OF GRAPH AND CPX *)
(  1)  357          NELT            : INTEGER;      (* NUMBER OF ELEMENTS IN THE SYMTAB *)
(  1)  358          NAME            : PACKED ARRAY[1..SYMSZE,1..10] OF CHAR;(* NAMES OF DESC *)
(  1)  359          PNO             : ARRAY[1..SYMSZE] OF INTEGER;(* DESC NO *)
(  1)  360          DPNO            : ARRAY[1..SYMSZE] OF INTEGER;(* DESC NO OF ASSOC DESC *)
(  1)  361          NARG            : ARRAY[-SYMSZE..SYMSZE] OF INTEGER;(* NUMBER OF ARGS *)
(  1)  362          VTYPE           : ARRAY[1..SYMSZE] OF 1..3;(*TYPE OF VAR - 1-NOMINAL,2-INT,3-STRU*)
(  1)  363          VCOST           : ARRAY[-SYMSZE..SYMSZE] OF INTEGER;(*COST OF EACH VARIABLE*)
(  1)  364          EVAL            : ARRAY[1..SYMSZE] OF INTEGER;(* NUMBER OF VALUES IN EXTND DOM*)
(  1)  365          MVAL            : ARRAY[1..SYMSZE] OF INTEGER;(* MINIMUM VALUE OF REF *)
(  1)  366          NVAL            : ARRAY[1..SYMSZE] OF INTEGER(* NUMBER OF VALUES *)
(  1)  367                        END;
(  1)  368       MSTR             = RECORD          (* METASELECTOR TABLE -- TWO ENTRIES FOR EACH FUNCTION&VALUE *)
(  1)  369          PNO             : PACKED ARRAY [1..GSIZE] OF 0..SYMSZE;(* FUNCTION ASSOCIATED WITH THIS META SELETOR (MS) *)
(  1)  370          VAL             : PACKED ARRAY [1..GSIZE] OF 0..MNVAL;(* VALUE OF THIS MS *)
(  1)  371          SYMPTR          : PACKED ARRAY [1..GSIZE] OF 0..SYMSZE;(* SYMBOL TABLE ENTRY FOR THIS MS (#PT OR FORALL) *)
(  1)  372          VARPTR          : PACKED ARRAY [1..GSIZE] OF 0..SYMSZE;(*POINTER INTO SYMTAB
(  1)  373                                                      OF THE DUMMY VARIABLE USED BY
(  1)  374                                                      THE UNARY FUNCTION UPON WHICH THE
(  1)  375                                                      METASELECTOR IS BASED*)
(  1)  376          PTR             : PACKED ARRAY[1..GSIZE] OF 0..GSIZE;(* ARRAY USED TO SORT THE MS TABLE *)
(  1)  377          F1COV           : PACKED ARRAY [1..GSIZE] OF INTEGER;(* NUMBER OF RULES IN F1 COVERED BY THIS MS *)
(  1)  378          FOCOV           : PACKED ARRAY[1..GSIZE] OF INTEGER;(* NUMBER OF RULES IN FO COVERED BY THIS MS *)
(  1)  379          METATRIM        : INTEGER;      (* PARAMETER= NUMBER OF MS'S TO USE IN PROGRAM *)
(  1)  380          NMST            : INTEGER       (* NUMBER OF METASELECTORS IN TABLE *)
(  1)  381                        END;
(  1)  382
(  1)  383
(  1)  384
(  1)  385       AQPARM           = RECORD          (* PARAMETERS FOR VL1 PART OF PROGRAM *)
(  1)  386          NVAR            : INTEGER;(* NUMBER OF VARIABLES IN AQ PROC *)
(  1)  387          CSTF            : ARRAY[1..MAXCSTF] OF INTEGER;(* COST FUNCTION A LIST *)
(  1)  388          TOLER           : ARRAY[1..MAXCSTF] OF REAL;(* TOLERANCE LIST *)
(  1)  389          NF              : INTEGER;(* NUMBER OF COST FORMULAS TO BE USED *)
(  1)  390          FREEC           : CPTR;(* POINTER TO FREE COMPLEX LIST *)
(  1)  391          SLOC            : ARRAY[1..GSIZE] OF INTEGER;(* LOCATION IN THE STABLE OF VBL*)
(  1)  392          CUTF1           : INTEGER;(* NUMBER OF F1 TO CHECK IN AQ *)
(  1)  393          LQST            : BOOLEAN;    (* IF TRUE, TRIM REFERENCES TO MINIMUM NEEDED *)
(  1)  394          MAXSTARAQ       : INTEGER(* MAXSTAR PARM IN AQ ALG *)
(  1)  395                        END;
(  1)  396       PARM             = RECORD          (* PARAMETERS FOR THE VL21 PART OF PROGRAM *)
(  1)  397          CSTF            : ARRAY[1..MAXCSTF] OF INTEGER;(*VL21 COST FUNCTIONS*)
(  1)  398          TOLER           : ARRAY[1..MAXCSTF] OF REAL;(*VL21 TOLERANCE*)
(  1)  399          NF              : INTEGER;(*NUMBER OF COST FUNCTIONS*)
(  1)  400          MAXSTAR         : INTEGER;(*MAXSTAR PARAMETER*)
(  1)  401          ALTER           : INTEGER;(* NUMBER OF ALTERNATIVES *)
(  1)  402          EXTMTY          : BOOLEAN;      (* ADD MST- LST- TYPE SELECTORS *)
(  1)  403          EQUIV           : BOOLEAN;      (* ADD =SAME TYPE SELECTORS *)
(  1)  404          NCONSIST        : INTEGER;(* NUMBER OF CONSISTENT ALTERNS TO GENERATE*)
(  1)  405          DESCTYPE        : (CHARACTERISTIC, DISCRIMINANT);(* TYPE OF GENERALIZATION *)
(  1)  406          MINCOVER        : INTEGER       (* MIN % OF F1 THAT MUST BE COVERED BY A RULE IN CHARACTERISTIC MODE *)
```

```
( 1)  407                          END;
( 1)  408          CARRAY          = ARRAY[1..101] OF CHAR;(* CHARACTER ARRAY -- MISC USES *)
( 1)  409          IARRAY          = ARRAY[0..MNVAL] OF INTEGER;(* INTEGER ARRAY -- MISC USES *)
( 1)  410          DSTRUC          = RECORD          (* DOMAIN STRUCTURE TABLE *)
( 1)  411             PREM           : ARRAY[1..NDES] OF VALTP;(* PREMISE OF DESC STRUCTURE RULE *)
( 1)  412             CONS           : ARRAY[1..NDES] OF VALTP;(* CONSEQUENCE OF DESC STRUCTURE RULE *)
( 1)  413             PNO            : ARRAY[1..NDES] OF INTEGER;(* POINTER TO SYMBOLTABLE *)
( 1)  414             NELE           : INTEGER(* NUMBER OF ELEMENTS IN THIS STRUCTURE USED SO FAR*)
( 1)  415                           END;
( 1)  416
( 1)  417
( 1)  418
( 1)  419          GSAR            = ARRAY[1..5] OF GPTR;
( 1)  420
( 1)  421        (*  VARIABLE DEFINITIONS       *)
( 1)  422
( 1)  423   VAR
( 1)  424          CHRR,CHRR1      : CHAR;           (* CURRENT COMMAND CHARACTERS *)
( 1)  425          I,J,K           : INTEGER;        (* MISC INTEGERS *)
( 1)  426          ES              : INTEGER;        (* EVENT SET BEING COVERED *)
( 1)  427          NINSTR          : INTEGER;        (* NUMBER OF GSTRUCT IN VL21 STAR *)
( 1)  428          INFILE          : INTEGER;        (* 0 READ FROM IFILE,  1 READ FROM CFILE *)
( 1)  429          ERR             : INTEGER;        (* ERROR FLAG FOR VLINT *)
( 1)  430          NMQ             : INTEGER;        (* NUMBER OF RULES IN MQ STAR *)
( 1)  431          DST             : DSTRUC;         (* DOMAIN STRUCTURES *)
( 1)  432          ARITHHEAD       : ARITHPTR;       (* HEAD OF ARITH STACKS TO APPLY TO RULES *)
( 1)  433          ARITHTEMP       : ARITHPTR;       (* TEMP USED TO PROCESS ARITH STACK S *)
( 1)  434          FREEG           : GPTR;           (* LIST OF FREE GRAPH STRUCTURES *)
( 1)  435          FREECPXTAB      : CPXTABPTR;          (* LIST OF CPXTABLE RECORDS NOT IN USE *)
( 1)  436          G,G1,G2         : GPTR;           (* MISC GPTRS *)
( 1)  437          STAR            : GPTR;           (* THE MAIN STAR *)
( 1)  438          RESTLIST        : GPTR;           (* LIST OF RESTRICTIONS *)
( 1)  439          R               : GPTR;           (* ?? *)
( 1)  440          COVSET          : GPTR;           (* F1 LIST *)
( 1)  441          GSET            : GPTR;           (* MAIN RULE BASE -- LINKED LIST *)
( 1)  442          MQ              : GPTR;           (* MQ STAR *)
( 1)  443          PSTAR           : GPTR;           (* PARTIAL STAR *)
( 1)  444          OPSTAR          : GPTR;           (* O STAR *)
( 1)  445          MST             : MSTR;         (* META SELECTOR TABLE *)
( 1)  446          NF1             : INTEGER;        (* NUMBER OF RULES IN F1 *)
( 1)  447          CRULENO         : INTEGER;        (* CURRENT RULE NUMBER -- USED TO ASSIGN NEXT NUMBER *)
( 1)  448          NEWTRACE        : INTEGER;        (* ?? *)
( 1)  449          NTIMES          : INTEGER;        (* ?? *)
( 1)  450          CONTWH          : BOOLEAN;        (* A FLAG USED IN CASE 'D' *)
( 1)  451          PTBL            : PT;             (* PARSE TABLE *)
( 1)  452          S               : SYMTAB;         (* SYMBOL TABLE *)
( 1)  453          STP,TRACE       : SET OF 1..10;   (* STOP POINTS AND TRACE POINTS *)
( 1)  454          PRULE           : BOOLEAN;        (* IF TRUE, PRINT RULES OUT *)
( 1)  455          FIXIT           : VALTP;
( 1)  456          AQP             : AQPARM;         (* AQ PARAMETERS TABLE *)
( 1)  457          CNSTCY          : ARRAY[1..GSIZE] OF INTEGER;(*CONSISTENCY VALUES*)
( 1)  458          PRM             : PARM;           (* VL21 PARAMETER TABLE *)
( 1)  459
( 1)  460              (* FILES USED IN THE PROGRAM *)
( 1)  461
( 1)  462          STAB            : FILE OF SYMTAB; (* SYMBOL TABLE SAVE FILE *)
( 1)  463          VL1EVE          : FILE OF CHAR;   (* VL1 EVENT FILE FOR VL1 MODE *)
( 1)  464          IFILE           : (*  INTERACTIVE *)
```

```
        FILE OF CHAR;
        GFILE          : FILE OF GRAPH;  (* RULE BASE SAVE FILE *)
        TABLES         : FILE OF CHAR;   (* PARSE TABLE FILE *)
        CFILE          : FILE OF CHAR;   (* FILE OF COMMANDS AND RULES *)
        EXPLAIN        : FILE OF CHAR;   (* FILE OF EXPLANATORY TEXT *)
        DFILE          : FILE OF DSTRUC; (* DOMAIN STRUCTURE SAVE FILE *)
  (*1
                 I N D U C E - 1  --  FORWARD PROCEDURE DEFINITIONS
  *)
*CALL PGRAPHA
*CALL ENTERPA
*CALL VLINTA
  (*1
                 I N D U C E - 1  --  DUMPG
  *)
*CALL DUMPG
  (*1
                 I N D U C E - 1  -- DUMPS
  *)
*CALL DUMPS
  (*1                I N D U C E - 1  --  DUMPARITH
  *)
*CALL DUMPARITH
  (*1
                 I N D U C E - 1  --  INSIDE
    *)
   (*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                              INSIDE(DNUM:INTEGER;V1,V2:VALTP):BOOLEAN;
          DETERMINES IF ONE SET, V2 IS A GENERALIZATION OF THE SET V1.  IF EVAL
      AND NVAL ARE THE SAME, THEN THE DOMAIN IS ASSUMED TO BE STRUCTURED
      OTHERWISE, IT IS CARTESIAN.  IF STRUCTURED, THEN THE STRUCTURE DSTRUC IS
      SEARCHED FOR POSSIBLE GENERALIZATIONS.
  @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*)
*CALL INSIDE
  (*1
               I N D U C E - 1  -- ADDSEL
  *)
  (*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                              ADDSEL(G:GPTR);
          ADD SYMMETRIC SELECTORS TO THE G STRUCT.  FIND ALL SELECTORS WHICH
      INVOLVE THE SAME FUNCTION AND SAME REFERENCE.  FORM A NEW SELECTOR
      WHICH IS LINKED TO ALL THESE.  THE PNO OF THE NEW SELECTOR
      IS THE NEGATIVE OF THE PNO OF THE ORIGINAL SELETORS.  VBL IS
      SET TO FALSE IN THESE SELECTORS AND ORDIRR IS ALSO SET TO TRUE.
      THEREFORE, SUBSETS OF ARGUMENTS CAN BE COMPARED USING SUBG1.
  @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*)
*CALL ADDSEL
  (*1
                 I N D U C E - 1  --  EXTND
  *)
  (*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                              EXTND(DNUM:INTEGER;V1,V2:VALTP);
          FIND THE EXTENSION OF V1 AGAINST V2, PUT RESULT IN RSLT.  DNUM IS THE
      DESCRIPTOR NUMBER (LOC IN STAB).  IF EVAL = NVAL, THEN THE DOMAIN
```

```
         IS ASSUMED TO BE CARTESIAN, OTHERWISE, THE DSTRUC RECORDS ARE
         SEARCHED TO FIND THE GENERALIZATION.  DSTRUC RECORDS ARE ASSUMED
         TO BE IN THE ORDER: LOWEST LEVEL GENERALIZATION FIRST.
     @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*)
*CALL EXTND
 (*1
                I N D U C E - 1  --  ILINE
 *)
 (*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                               ILINE;
    INPUT LINE OF INFORMATION FROM TTY OR CFILE DEPENDING ON INFILE
 @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*)
*CALL ILINE
      (*  ILINE  *)
 (*1
                I N D U C E - 1  --  PEOS AND GETCHRR
 *)
 (*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                               PEOS(I:INTEGER):BOOLEAN;
         DETERMINE IF AT THE END OF SEGMENT OR LINE
 @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*)
*CALL PEOS
 (*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                               GETCHRR(VAR C:CHAR);
         GET CHARACTER FROM INPUT FILE
 @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*)

*CALL GETCHRR
 (*1
                I N D U C E - 1  --  INIT
 *)
 (*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                               INIT(
         INTIALIZE CERTAIN PARAMETERS, READ IN SYMBOL TABLE AND PARSE TABLE
    FROM STAB AND TABLES.
 @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*)
*CALL INIT
 (*1
              I N D U C E - 1  --  NEWG, GIN, AND GOUT
 *)
 (*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                            NEWG
 @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*)
*CALL NEWG
 (*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                               GIN(G:GPTR);
         INPUT GRAPH STRUCTURE
 @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*)
*CALL GIN
 (*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                               GOUT(G:GPTR);
         OUTPUT GRAPH STRUCTURE
 @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*)
*CALL GOUT
 (*             I N D U C E - 1  --  NEWCPXTAB
 *)
*CALL NEWCPXTAB
 (*1
```

```
                     I N D U C E - 1   --  FREETAB, NEWC, FREECPX
 *)
*CALL FREETAB


*CALL NEWC


*CALL FREECPX
 (*1
                     I N D U C E - 1   --  EXPLN
 *)
 (*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                                         EXPLN
 @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*)
*CALL EXPLN
 (*1
                     I N D U C E - 1   --  PMETAD
 *)
 (*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                                        PRINT METAD
 @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*)
*CALL PMETAD
 (*1
                     I N D U C E - 1   --  ENTERP
 *)
 (*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                             ENTERP
 @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*)
*CALL ENTERP
 (*1
                     I N D U C E - 1   --  SOUT AND ADDCONS
 *)
 (*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                                          SOUT;
                OUTPUT SYMBOL TABLE ON STAB
 @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*)
*CALL SOUT
 (*
                     I N D U C E - 1   --  ADDCONS

 *)
 (*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@


                                                   ADDCONS(G1,G2:GPTR);

                  ADD CONSEQUENCE OF RESTRICTION TO GRAPH IF NOT ALREADY THERE


 @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*)
*CALL ADDCONS
 (*1
                     I N D U C E - 1   --  ALLC
 *)
 (*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

```
(  1)  639                                ALLC(VAR F1:CPTR;
(  1)  640           TRANSLATE FROM GRAPH STRUCTURE INTO COMPLEX FOR AQ
(  1)  641    @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*)
(  1)  642   *CALL ALLC
(  1)  643    (*1
(  1)  644                  I N D U C E - 1  --  CALCARITH
(  1)  645    *)
(  1)  646   *CALL CALCARITH
(  1)  647    (*1
(  1)  648                  I N D U C E - 1  --  SUBG1
(  1)  649    *)
(  1)  650    (*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
(  1)  651             DETERMINE IF G1 IS SUBGRAPH OF G2.  SUBG1 FINDSTHE  FIRST MATCHING
(  1)  652    SELECTORS OF G1 AND G2.  SUBG IS CALLED TO MATCH THE REST OF
(  1)  653    THE GRAPH.  A BACKTRACK ARRAY LIST IS MAINTAINED TO FACILITATE
(  1)  654    BACKING UP IN THE SPANNING TREE IF TWO NODES DONT MATCH.  LIST
(  1)  655    CONTAINS THE NODE NUMBER IN G1, AND THE LINK NUMBERS IN G1 AND
(  1)  656    G2 FOR EACH MATCH.  AT EACH MATCH, ASSGN IS SET TO SPECIFY THE
(  1)  657    1-TO-1 CORRESPONDENCE WHICH EXISTS BETWEEN GRAPHS.
(  1)  658    @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*)
(  1)  659   *CALL SUBG1
(  1)  660    (*1
(  1)  661                  I N D U C E - 1  --  PCPX
(  1)  662    *)
(  1)  663    (*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
(  1)  664                                    PCPX(F:CPTR);
(  1)  665    @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*)
(  1)  666   *CALL PCPX
(  1)  667    (*1
(  1)  668                  I N D U C E - 1  --  AQ
(  1)  669    *)
(  1)  670       (*$D-     TURN OFF DEBUGGING FOR AQ--ITS TOO BIG *)
(  1)  671    (*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
(  1)  672                                  AQ(GSUB:GPTR;VL1M:BOOLEAN;F1,F2:CPTR;
(  1)  673             ACTUAL AQ ALGORITHM -- MUCH LIKE AQ7
(  1)  674    @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*)
(  1)  675   *CALL AQ
(  1)  676    (*DEBUGOFF  TURN ON DEBUGGING AGAIN
(  1)  677   1
(  1)  678                  I N D U C E - 1  --  CLEANCPX
(  1)  679    *)
(  1)  680   *CALL CLEANCPX
(  1)  681    (*1
(  1)  682                  I N D U C E - 1  --  LQST2
(  1)  683    *)
(  1)  684   *CALL LQST2
(  1)  685    (*1
(  1)  686                   I N D U C E - 1  --  AQSET
(  1)  687    *)
(  1)  688    (*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
(  1)  689                                  AQSET(GSET:GPTR;
(  1)  690             SETS UP STRUCTURES FOR AQ PROCEDURE.  CVAL CONTAINS BIT POS
(  1)  691          REPRESENTATION OF REFERENCES IN GSUB.  SLOC CONTAINS
(  1)  692          POINTER TO SYMBOL TABLE LOCATION OF ASSOCIATED DESCRIPTOR
(  1)  693    @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*)
(  1)  694   *CALL AQSET
(  1)  695    (*1
(  1)  696                  I N D U C E - 1  --  ENTERD
```

```
*)
(*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                                    ENTERD
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*)
*CALL ENTERD
(*1
               I N D U C E - 1  --  VL1
*)
(*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                                    VL1
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*)
*CALL VL1
(*1
               I N D U C E - 1  --  NEWGP
*)
(*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                                    NEWGP(GN,G1:GPTR;
          FIND A NEW GRAPH WITH MNODE SELECTORS IN IT.
     COUNT IN G1 RECORDS THE NUMBER OF TIMES WHICH A SELECTOR HAS
     BEEN USED IN PREVIOUS GRAPHS.  COUNT IN GN INDICATES THE NUMBER
     OF OCCURRENCES OF THIS VBL IN THE NEW GP
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*)
*CALL NEWGP
(*1
               I N D U C E - 1  --  PGRAPH
*)
(*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                                    PGRAPH;
     PRINTS A VL21 FORMULA
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*)
*CALL PGRAPH
(*1
               I N D U C E - 1  --  TOKEN
*)
(*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                                    TOKEN(
     FINDS THE NEXT TOKEN IN THE INPUT STREAM
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*)
*CALL TOKEN
(*1
             I N D U C E - 1  --  VLINT
*)
(*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                                    VLINT(
          PARSE A VL21 EXPRESSION AND PERFORM SEMANTIC ACTIONS AS REQUIRED
      TO FORM A GRAPH.  ADD ENTRIES TO SYMBOL TABLE AND GRAPH STRUCTURE.
      PSTK IS THE STACK OF NONTERMINALS TRIED ALREADY
      VSTK IS A STACK OF VALUE SETS FOR REFERENCES
      FSTK IS A STACK OF DESCRIPTORS AND DUMMY VARIABLES
      SSTK IS THE TOP DOWN PARSE OF THE EXPRESSION SO FAR
      EACH TOKEN FROM THE TOKEN ROUTINE IS MATCHED WITH AN ELEMENT
      IN A ROW OF THE PARSE TABLE (IN THIS TABLE, POS NUMBERS ARE
      TERMINALS, NEG ARE NONTERMINALS, POS NUMBERS MATCH THE NUMBER
      RETURNED BY TOKEN, NEG NUMBERS SPECIFY WHICH ROW OF THE PARSE
      TABLE TO PARSE NEXT).
           IF AN ELEMENT MATCHES, IT IS PLACED ON SSTK, IF IT IS AT
      THE END OF A ROW (PRODUCTION), THEN THE ELEMENTS OF SSTK ARE REPLACED
      BY -ROW' OF THE MATCH, PSTK IS POPPED AND THE CURRENT ROW IS THE
```

```
       TOP ELEMENT OF PSTK (ALONG WITH THE COLUMN POINTER IN PSTK).
       WHEN YOU GETTO THE BOTTOM OF PSTK, THEN YOU^RE DONE.
 @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*)
*CALL VLINT
 (*1
              I N D U C E - 1  --  COSTG
 *)
 (*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                              COSTG(P:GPTR;
             EVALUATE THE COST OF THIS GRAPH (COST FUNCTION CT).
 @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*)
*CALL COSTG
 (*1
             I N D U C E - 1  --  ADDTOMQ
 *)
*CALL ADDTOMQ
 (*1
              I N D U C E - 1  --  TRIMG
 *)
 (*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                              TRIMG(VAR NSTAR:GPTR;
            TRIM A LIST OF GRAPHS TO MAXS GRAPHS ACCORDING TO FUNCTIONAL
 @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*)
*CALL TRIMG
 (*1
              I N D U C E - 1  --  COMPMS
 *)
 (*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                          COMPMS
 @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*)
*CALL COMPMS
 (*1
              I N D U C E - 1  --  TRIMM
 *)
 (*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                         TRIMM
 @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*)
*CALL TRIMM
 (*1
              I N D U C E - 1  --  ADDMETA
 *)
 (*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                        ADDMETA
 @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*)
*CALL ADDMETA
 (*1
              I N D U C E - 1  --  PROCARITH
 *)
*CALL PROCARITH
 (*1
              I N D U C E - 1  --  ADDML
 *)
 (*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                         ADDML
      ADD THE MOST AND LEAST PARTS OF 2-ARY FNCTNS
 @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*)
*CALL ADDML
 (*1
```

```
              I N D U C E - 1  --  COVER
   *)
   (*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                                  COVER(ES:INTEGER);
   @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*)
  *CALL COVER
   (*COVER*)

   (*1
              I N D U C E - 1  --  MAIN PROGRAM
   *)
  BEGIN
       INIT;     LINELIMIT(OUTPUT,-1);      RESET(GFILE);
       WHILE NOT EOF(GFILE) DO BEGIN
          NEWG(G);     GIN(G);     G^.NXTN:=GSET;     GSET:=G;
          END;(*WHILE*)

       CHRR:=' ';     INFILE:=1;     RESET(CFILE);
       WRITELN(OUTPUT,'ENTER ONE CHAR: (P)ARAMETERS, (V)L1, (C)OVER, ', '(M)ODIFY, (H)ELP FOR MORE OPTNS');
       REPEAT
          IF INFILE=0 THEN WRITELN(OUTPUT,'ENTER ONE CHAR: (P)ARAMETERS, (V)L1, (C)OVER, ',
             '(M)ODIFY, (H)ELP FOR MORE OPTNS');
          ILINE;
          REPEAT
             GETCHRR(CHRR)              (* SKIP LEADING BLANKS *)
             UNTIL CHRR<>' ';
          IF CHRR IN ['R','M','C','P','E','L','H','V','S','D','A','N','Z','X'] THEN CASE CHRR OF(* PROCESS COMMAND *)
             'H':BEGIN
                WRITELN(OUTPUT,'     READ IN RESTRICTIONS (R) ');     WRITELN(OUTPUT,'     MODIFY RULES (M) ');
                WRITELN(OUTPUT,'     GET HELP (H) ');
                WRITELN(OUTPUT,'     INCLUDE MOST-LEAST TYPE SELECTORS (L)');
                WRITELN(OUTPUT,'     COVER SET OF RULES (C) ');     WRITELN(OUTPUT,'     USE VL1 MODE (V) ');
                WRITELN(OUTPUT,'     MODIFY PARAMETERS (P) ');
                WRITELN(OUTPUT,'     ENTER DOMAIN STRUCTURE RULES (E)');
                WRITELN(OUTPUT,'     ADD EQUIV TYPE SEL (S) ');
                WRITELN(OUTPUT,'     ENTER ARITHMETIC FUNCTIONS (A) ');
                WRITELN(OUTPUT,'     ADD ARITHMETIC FUNCTIONS TO RULE BASE (N) ');
                WRITELN(OUTPUT,'     READ IN SYMBOLIC CONCEPT AND SUBSTITUTE IT INTO RULES (X)');
                WRITELN(OUTPUT,'     QUIT (Q) ');
                IF PEOS(I) THEN CHRR:='H' ELSE WHILE NOT PEOS(I) DO GETCHRR(CHRR);
                IF CHRR IN['R','M','C','P','L','E','V','S','A','N','X'] THEN(* HELP CASES *)
                CASE CHRR OF
                   'R': EXPLN(21);
                   'M': EXPLN(22);
                   'C': EXPLN(23);
                   'P': EXPLN(24);
                   'V': EXPLN(27);
                   'E': EXPLN(25);
                   'S': EXPLN(29);
                   'L': EXPLN(28);
                   'A': EXPLN(30);
                   'N': EXPLN(31);
                   'X': EXPLN(32);
                   END(*CASE STMT*)

                ELSE EXPLN(26);
                WRITELN(OUTPUT);
                END;
```

```
(  1)  871              'P':BEGIN                    (* DISPLAY AND CHANGE PARAMETERS *)
(  1)  872                 ENTERP;
(  1)  873                 END;(*CASE P*)
(  1)  874
(  1)  875              'E':ENTERD;                  (* ENTER DOMAIN DESCRIPTIONS *)
(  1)  876              'V':VL1;                     (* ENTER VL1 MODE *)
(  1)  877              'L':BEGIN                    (* ADD MST-/LST- TYPE DESCRIPTORS *)
(  1)  878                 ADDML;       PRM.EXTMTY:=TRUE;
(  1)  879                 END;
(  1)  880              'S':BEGIN                    (* ADD =SAME TYPE SELECTORS *)
(  1)  881                 G:=GSET;
(  1)  882                 WHILE G<>NIL DO BEGIN
(  1)  883                    ADDSEL(G);       G:=G^.NXTN;
(  1)  884                    END;
(  1)  885                 PRM.EQUIV:=TRUE;
(  1)  886                 END;
(  1)  887              'R':BEGIN                    (* READ IN RESTRICTIONS *)
(  1)  888                 NEWG(G);       VLINT(G,ERR,ES,VLRULE,NIL);       G^.NXTN:=RESTLIST;       RESTLIST:=G;
(  1)  889                 END;(*CASE R*)
(  1)  890
(  1)  891              'Z': BEGIN(* DUMP ARITH STUFF FOR DEBUGGIN *)
(  1)  892                 ARITHTEMP:=ARITHHEAD;
(  1)  893                 WHILE ARITHTEMP<>NIL DO BEGIN
(  1)  894                    DUMPARITH(ARITHTEMP);       ARITHTEMP:=ARITHTEMP^.NEXT;
(  1)  895                    END;
(  1)  896                 END;(* CASE Z *)
(  1)  897              'D': BEGIN(* DUMP GRAPHS FOR DEBUGGING *)
(  1)  898                 G:=GSET;
(  1)  899                 WHILE G<>NIL DO BEGIN
(  1)  900                    DUMPG(G);       G:=G^.NXTN;
(  1)  901                    END;(*WHILE G*)
(  1)  902                 DUMPS;
(  1)  903                 END;(*CASE D*)
(  1)  904
(  1)  905              'A': BEGIN                   (* ADD A NEW ARITHMETIC FUNCTION *)
(  1)  906                 ARITHTEMP := ARITHHEAD;       NEW(ARITHHEAD);(* GET A NEW ARITHSTACK *)
(  1)  907                 ARITHHEAD^.NEXT:=ARITHTEMP;(* ADD TO LINKED LIST *)
(  1)  908                 ARITHHEAD^.SIZE:=0;       (* EMPTY *)
(  1)  909                 ARITHHEAD^.APPLIED:=FALSE;(* NOT YET APPLIED VIA 'N' COMMAND *)
(  1)  910                 ERR:=1;
(  1)  911                 IF INFILE=0 THEN WRITELN(OUTPUT,'ENTER:  FUNCTION(ARGS)=ARITHMETIC EXPRESSION');
(  1)  912                 WHILE ERR<>0 DO VLINT(NIL,ERR,ES,ARITHDD,ARITHHEAD);
(  1)  913                 END;(* CASE A *)
(  1)  914
(  1)  915              'N': BEGIN                   (* PROCESS THE ARITHMETIC DERIVED DESCRIPTORS *)
(  1)  916                 ARITHTEMP:=ARITHHEAD;    (* LOOP THRU DERIVED DESCRIPTORS *)
(  1)  917                 WHILE ARITHTEMP<>NIL DO BEGIN
(  1)  918                    PROCARITH(ARITHTEMP,GSET);       ARITHTEMP:=ARITHTEMP^.NEXT;(* LOOP *)
(  1)  919                    END;(* WHILE *)
(  1)  920                 END;(*CASE N *)
(  1)  921
(  1)  922              'M':BEGIN                    (* MODIFY RULE BASE *)
(  1)  923                 IF INFILE=0 THEN WRITELN(OUTPUT,'ADD OR DELETE RULE? ');
(  1)  924                 ILINE;       GETCHRR(CHRR1);
(  1)  925                 IF CHRR1 IN ['A','D'] THEN CASE CHRR1 OF
(  1)  926                    'A': BEGIN             (* ADD A NEW RULE *)
(  1)  927                       NEWG(G);       ERR:=1;       NEWC(G^.MSEL);
(  1)  928                       WHILE ERR<>0 DO BEGIN
```

```
( 1)  929                        ERR:=0;
( 1)  930                        IF INFILE=0 THEN WRITELN(OUTPUT,'ENTER RULE');
( 1)  931                        VLINT(G,ERR,ES,VLRULE,NIL);(* READ AND CONVERT TO GRAPH G *)
( 1)  932                        END;(*WHILE*)
( 1)  933
( 1)  934                     G^.NXTN:=GSET;       R:=RESTLIST;
( 1)  935                     WHILE R<>NIL DO BEGIN
( 1)  936                        IF SUBG1(R,G,1,AQP.FREEC,TRUE,NIL) THEN ;
( 1)  937                        R:=R^.NXTN;     (* ADD RESTRICTIONS TO RULES *)
( 1)  938                        END;
( 1)  939                     GSET:=G;
( 1)  940                     END;(*CASE A*)
( 1)  941
( 1)  942                  'D': BEGIN            (* DELETE A RULE *)
( 1)  943                     G1:=GSET;       CONTWH := TRUE;
( 1)  944                     WHILE (G1<>NIL) AND CONTWH DO BEGIN
( 1)  945                        WRITELN(OUTPUT,'DELETE THE FOLLOWING RULE? ');       PGRAPH(G1,S);       ILINE;
( 1)  946                        GETCHRR(CHRR);
( 1)  947                        IF CHRR = 'Y' THEN BEGIN
( 1)  948                           G2:=G1^.NXTN;       G1^.NXTN:=FREEG;       FREEG:=G1;
( 1)  949                           IF G1=GSET THEN GSET:=G2 ELSE G^.NXTN:=G2;
( 1)  950                           G1:=G2;
( 1)  951                           END;
( 1)  952                        IF CHRR = 'N' THEN BEGIN
( 1)  953                           G:=G1;       G1:=G1^.NXTN;
( 1)  954                           END;
( 1)  955                        IF CHRR = 'Q' THEN CONTWH := FALSE;
( 1)  956                        END;(* WHILE *)
( 1)  957                     END(* CASE D *)
( 1)  958                  END;(*CASE STMT *)
( 1)  959
( 1)  960               END;(*CASE M*)
( 1)  961
( 1)  962            'C':BEGIN
( 1)  963               COVER(ES);
( 1)  964               END;(*CASE C*)
( 1)  965
( 1)  966            'X': BEGIN
( 1)  967               CHRR:='R';(*TELL VLINT TO DO RESTRICTIONS *)
( 1)  968               NEWG(G); (* GET A NEW GRAPH *)
( 1)  969               REPEAT VLINT(G,ERR,ES,VLRULE,NIL) UNTIL ERR=0;
( 1)  970               CHRR:='X';(* TELL ADDCONS TO DO EXCHANGE *)
( 1)  971               G1:=GSET;(* GO THRU RULE BASE *)
( 1)  972               WHILE G1<>NIL DO BEGIN
( 1)  973                  IF SUBG1(G,G1,1,AQP.FREEC,TRUE,NIL) THEN;
( 1)  974                  G1:=G1^.NXTN;(* SUBG1/ADDCONS WILL HANDLE IT *)
( 1)  975                  END;
( 1)  976               END;(* CASE X *)
( 1)  977
( 1)  978
( 1)  979            END;(* CASE STMT *)
( 1)  980
( 1)  981         UNTIL CHRR='Q'
( 1)  982       END.
( 1)  983       (* PROGRAM VL21 *)
( 2)    1 *DECK PGRAPHA
( 2)    2 PROCEDURE PGRAPH
( 2)    3      (G               : GPTR;
```

```
( 2)  4        S              : SYMTAB);
( 2)  5        FORWARD;                                (* PRINT GRAPHS AS VL21 EXPRESSIONS *)
( 3)  1  *DECK ENTERPA
( 3)  2  PROCEDURE ENTERP;
( 3)  3        FORWARD;                          (* ENTER PARAMETERS *)
( 4)  1  *DECK VLINTA
( 4)  2  PROCEDURE VLINT
( 4)  3       (G              : GPTR;
( 4)  4        VAR ERR        : INTEGER;
( 4)  5        VAR ES         : INTEGER;
( 4)  6        PRODNUM        : INTEGER;
( 4)  7        A              : ARITHPTR);      (* GENERAL RULE PARSER AND EXPRESSION PARSER *)
( 4)  8        FORWARD;
( 5)  1  *DECK DUMPG
( 5)  2  PROCEDURE DUMPG
( 5)  3       (G              : GPTR);(*     PURPOSE:  TO DUMP THE INFORMATION IN A PARTICULAR GSTRUC
( 5)  4   *)
( 5)  5  VAR
( 5)  6        I,J            : INTEGER;
( 5)  7
( 5)  8  BEGIN
( 5)  9        WITH G^ DO BEGIN
( 5) 10           WRITELN(OUTPUT, 'G^',ORD(G),' RNO:  ',RNO:3, ' FP:',FP:1, ' MSEL: ',ORD(MSEL), ' COST: ',COST[1]:3,COST[2]:3,
( 5) 11              COST[3]:3,COST[4]:3);
( 5) 12           FOR I:= 1 TO MNVAL DO IF I IN ESET THEN WRITE(OUTPUT,I:3);
( 5) 13           WRITE(OUTPUT,' NXTN:',ORD(NXTN),' NNEG:',ORD(NNEG));      WRITELN(OUTPUT);      I:=1;
( 5) 14           WHILE PNO[I]<>0 DO BEGIN
( 5) 15              WRITE(OUTPUT,' INDEX ',I:2,' VBL:  ',VBL[I]:1,' ORDIRR',ORDIRR[I]:1, 'VAL: ');
( 5) 16              FOR J:= 0 TO MNVAL DO IF J IN VAL[I] THEN WRITE(OUTPUT,J:3);
( 5) 17              WRITELN(OUTPUT);
( 5) 18              WRITE(OUTPUT,' COUNT ',COUNT[I]:3,' ASSGN ',ASSGN[I]:3, ' PNO: ',PNO[I]:3,' DUMNUM ',DUMNUM[I]:3);
( 5) 19              WRITELN(OUTPUT);
( 5) 20              WRITE(OUTPUT,'LNK: ');
( 5) 21              FOR J:= 1 TO MLNK DO WRITE(OUTPUT,LNK[I,J]:3);
( 5) 22              WRITELN(OUTPUT);      I:=I+1;
( 5) 23              END;(* WHILE I *)
( 5) 24           END; (*WITH*)
( 5) 25        END;(*DUMPG*)
( 6)  1  *DECK DUMPS
( 6)  2  PROCEDURE DUMPS;(*      PURPOSE:  TO  DUMP THE SYMBOL TABLE
( 6)  3   *)
( 6)  4  VAR
( 6)  5        I,J            : INTEGER;
( 6)  6
( 6)  7  BEGIN
( 6)  8        WRITELN(OUTPUT);      WRITELN(OUTPUT,'NELT ',S.NELT:3);
( 6)  9        FOR I:= 1 TO S.NELT DO BEGIN
( 6) 10           WRITE(OUTPUT,'NAME ');
( 6) 11           FOR J:=1 TO 10 DO WRITE(OUTPUT,S.NAME[I,J]);
( 6) 12           WRITE(OUTPUT,'  PNO ',S.PNO[I]:3,' DPNO ',S.DPNO[I]:3, ' NARG ',S.NARG[-1]:3,S.NARG[I]:3, ' VTYPE ',
( 6) 13              S.VTYPE[I]:1, ' VCOST ',S.VCOST[I]:2, ' EVAL ',S.EVAL[I]:2, ' MVAL ',S.MVAL[I]:2, ' NVAL ',S.NVAL[I]:2);
( 6) 14           WRITELN(OUTPUT);
( 6) 15           END; (*FOR I*)
( 6) 16        END;(*DUMPS*)
( 7)  1  *DECK DUMPARITH
( 7)  2  PROCEDURE DUMPARITH
( 7)  3       (A              : ARITHPTR);(* PURPOSE:  TO DUMP AN ARITH STACK TO TERMINAL
( 7)  4   *)
```

```
( 7)   5  VAR
( 7)   6        I,J             : INTEGER;
( 7)   7  BEGIN
( 7)   8        WITH A^ DO BEGIN
( 7)   9           FOR I:=1 TO SIZE DO BEGIN
( 7)  10              WRITE(OUTPUT,I:3,' ACTION:',ORD(STACK[I].ACTION):2,' ARGUMENT:',STACK[I].ARGUMENT:3, ' DUMMY:');       J:=1;
( 7)  11              WHILE STACK[I].DUMMY[J]<>0 DO BEGIN
( 7)  12                 WRITE(OUTPUT,STACK[I].DUMMY[J]:3);       J:=J+1;
( 7)  13                 END;(*WHILE*)
( 7)  14              WRITELN(OUTPUT);
( 7)  15              END;(* FOR I *)
( 7)  16           END;(*WITH*)
( 7)  17        END;(* DUMPARITH *)
( 8)   1  *DECK INSIDE
( 8)   2  FUNCTION INSIDE
( 8)   3       (DNUM            : INTEGER;
( 8)   4        V1,V2           : VALTP;
( 8)   5        INSD            : BOOLEAN): BOOLEAN;(*     PURPOSE:
( 8)   6         DNUM = DOMAIN NUMBER (POINTER INTO SYMBOL TABLE FOR DOMAIN )
( 8)   7         V1,V2 = THE VALUE SETS OF THE TWO RULES
( 8)   8         INSD = IF TRUE, V1 MUST BE A STRICT SUBSET OF V2
( 8)   9                IF FALSE, V1 MUST MERELY INTERSET V2
( 8)  10   *)
( 8)  11  VAR
( 8)  12        I,J             : INTEGER;
( 8)  13  BEGIN
( 8)  14        INSIDE:=FALSE;       DNUM:=ABS(DNUM);
( 8)  15        IF S_VTYPE[DNUM]<>3 THEN          (* IF NOMINAL OR INTERVAL... *)
( 8)  16           IF INSD AND (V1<=V2) OR (NOT INSD AND (V1 * V2 <>[])) THEN(* AND EITHER INSD AND V1 IS SUBSET OF V2 OR
( 8)  17               NOT INSD AND V1 INTERSECTS V2 *)
( 8)  18           INSIDE:=TRUE
( 8)  19           ELSE
( 8)  20        ELSE                              (* IT IS STRUCTURED DOMAIN *)
( 8)  21        WITH DST DO BEGIN
( 8)  22           FOR I:=NELE DOWNTO 1 DO IF DNUM=PNO[I] THEN(* IF THIS NODE POINTS TO US...*)
( 8)  23              IF CONS[I]<=V2 THEN         (* AND THE CONSEQUENCE IS A SUBSET OF OURS *)
( 8)  24              V2:=V2+PREM[I];             (* EXPAND V2 (LOCALLY) TO INCLUDE THE CONSEQUENCE *)
( 8)  25           IF INSD AND (V1<=V2) OR (NOT INSD AND (V1 * V2 <>[])) THEN INSIDE:=TRUE;
( 8)  26                                          (* NOW TRY TEST AGAIN AND IF TRUE, SET FLAG *)
( 8)  27           END;(*WITH*)
( 8)  28
( 8)  29        END;(* INSIDE *)
( 9)   1  *DECK ADDSEL
( 9)   2  PROCEDURE ADDSEL
( 9)   3       (G               : GPTR);
( 9)   4  LABEL
( 9)   5        99;                               (* JUMP OUT IF ERROR *)
( 9)   6  VAR
( 9)   7        LND,NND,I,J,K,L: INTEGER;
( 9)   8  BEGIN
( 9)   9        WITH G^ DO                        (* EXAMINE THIS GRAPH STRUCTURE*)
( 9)  10        BEGIN
( 9)  11           LND:=1;                        (* FIND FIRST EMPTY INDEX IN GRAPH *)
( 9)  12           WHILE LNK[LND,1]<>0 DO BEGIN   (* AND PUT IT IN LND *)
( 9)  13              LND:=LND+1;
( 9)  14              IF LND>GSIZE-1 THEN BEGIN
( 9)  15                 WRITELN(OUTPUT,'CANT ADD EQUIVALENCE', ' PREDICATES TO RULE--EXCEEDS GSIZE.');
( 9)  16                 GOTO 99;
```

```
( 9)  17              END;
( 9)  18            END; (* WHILE *)
( 9)  19          NND:=LND-1;                    (* NND IS LAST USED ENTRY IN GRAPH *)
( 9)  20          FOR I:=1 TO NND DO             (* USE COUNT AS A FLAG TO INDICATE THAT WE HAVE *)
( 9)  21          COUNT[I]:=0;                   (* PROCESSED THIS NODE.  INITIALIZE ALL NODES TO 0 *)
( 9)  22          FOR I:=1 TO NND DO             (* EXAMINE EACH NODE *)
( 9)  23          IF (COUNT[I]=0)AND(NOT VBL[I]) AND(LNK[I,2]=0)
( 9)  24             THEN BEGIN(* IF WE HAVEN'T PROCESSED THIS NODE AND ITS A UNARY PREDICATE *)
( 9)  25             K:=2;                       (* K WILL BE INDEX TO ALLOW COPYING LINKS *)
( 9)  26             LNK[LND,1]:=LNK[I,1];       (* COPY THE ONE AND ONLY LINK TO DUMMY VARIABLE *)
( 9)  27             FOR J:=I+1 TO NND DO        (* EXAMINE THE REMAINDER OF THE GRAPH *)
( 9)  28             IF (VAL[J]=VAL[I])AND(PNO[I]=PNO[J])
( 9)  29                THEN BEGIN(* IF THE VALUES ARE THE SAME AND THE DESCRIPTORS ARE THE SAME *)
( 9)  30                COUNT[J]:=1;             (* MARK THE NODE AS "PROCESSED" *)
( 9)  31                LNK[LND,K]:=LNK[J,1];       (* COPY ITS DUMMY VARIABLE LINK *)
( 9)  32                K:=K+1;                  (* INCREMENT OUR LINK COUNTER *)
( 9)  33                END;
( 9)  34             LNK[LND,K]:=0;              (* MAKE SURE OUR LINK LIST ENDS *)
( 9)  35             IF K<>2 THEN                (* IF WE HAVE AT LEAST 2 DUMMY VARIABLES *)
( 9)  36             BEGIN
( 9)  37                PNO[LND]:=-PNO[I];       (* COPY THE SYMBOL POINTER AND NEGATE IT*)
( 9)  38                S.VCOST[PNO[I]]:=S.VCOST[-PNO[I]];     (* COPY THE COST *)
( 9)  39                            (* USED TO SELECT EQUIV TYPE SELECTORS IN COVER SO
( 9)  40                            RT*)
( 9)  41
( 9)  42                VAL[LND]:=[0..MNVAL];      (* SET REFERENCE TO * *)
( 9)  43                VBL[LND]:=FALSE;         (* SET "NOT A VARIABLE" *)
( 9)  44                ORDIRR[LND]:=TRUE;       (* SET ORDER IRRELEVANT *)
( 9)  45                FOR J:=1 TO K-1 DO BEGIN(*ADD BACK POINTERS*)
( 9)  46
( 9)  47                   L:=1;                 (* FOR EACH VARIABLE THAT WE POINT TO *)
( 9)  48                   WHILE LNK[LNK[LND,J],L]<>0 DO BEGIN      (* EXAMINE ITS LINKS--FIND LAST ONE *)
( 9)  49                      L:=L+1;
( 9)  50                      IF L>MLNK-1 THEN BEGIN
( 9)  51                         WRITELN(OUTPUT,'CANT ADD EQUIVALENCE PREDICATES TO RULE--MLNK EXCEEDED');
( 9)  52                         END;
( 9)  53                      END;    (* WHILE *)
( 9)  54                   LNK[LNK[LND,J],L]:=LND;       (* MAKE LAST ONE POINT BACK TO US *)
( 9)  55                   LNK[LNK[LND,J],L+1]:=0;       (* MAKE THE NEXT ONE A ZERO *)
( 9)  56                   END;       (*FOR J*)
( 9)  57
( 9)  58                LND:=LND+1;              (* WE HAVE NOW OCCUPIED THIS LOCATION *)
( 9)  59                END     (*K<>2*)
( 9)  60
( 9)  61             ELSE LNK[LND,1]:=0;         (* IF WE ONLY FOUND ONE FUNCTION W/ THIS VALUE, FORGET IT *)
( 9)  62             END;(*---AND---*)
( 9)  63
( 9)  64          END;(*WHILE*)
( 9)  65
( 9)  66  99:    ;
( 9)  67          END;(* ADDSEL *)
( 10)  1  *DECK EXTND
( 10)  2  PROCEDURE EXTND
( 10)  3       (DNUM            : INTEGER;
( 10)  4          V1,V2         : VALTP);
( 10)  5  VAR
( 10)  6          I,J,LL        : INTEGER;
( 10)  7          TRSLT         : VALTP;          (* TEMPORARY RESULT *)
```

```
( 10)    8  BEGIN
( 10)    9        TRSLT:=V1;       DNUM:=ABS(DNUM); (* GET ABS VALUE OF RULE NUMBER *)
( 10)   10        CASE S.VTYPE[DNUM] OF
( 10)   11  1:       TRSLT:=[0..MNVAL]-V2;          (* IF NOMINAL, ITS EASY.  USE EVERYTHING EXCEPT V2 *)
( 10)   12  2:       BEGIN                          (* IF INTERVAL, ITS TRICKY,  FIND THE FIRST INTERVAL *)
( 10)   13              I:=0;
( 10)   14              WHILE (I<MNVAL)AND(NOT (I IN V1)) DO(* FIND LOWEST I IN V1 *)
( 10)   15              I:=I+1;       J:=0;
( 10)   16              WHILE (J<MNVAL)AND(NOT (J IN V2)) DO(* FIND LOWEST J IN V2 *)
( 10)   17              J:=J+1;
( 10)   18              IF I<J THEN                 (* IF I<J ADD ELEMENTS FROM I TO J-1 INTO V1 *)
( 10)   19              TRSLT:=TRSLT+ [I..J-1]
( 10)   20              ELSE                        (* IF I>= J ... *)
( 10)   21              BEGIN
( 10)   22                 WHILE (J<MNVAL)AND(J IN V2) DO (* SKIP ALL THE J'S IN V2 *)
( 10)   23                 J:=J+1;       TRSLT:=TRSLT+ [J..MNVAL];
( 10)   24                 END;
( 10)   25              END;(*CASE 2*)
( 10)   26
( 10)   27  3:       WITH DST DO                    (* IF STRUCTURED...*)
( 10)   28           BEGIN
( 10)   29              FOR I:=1 TO NELE DO         (* SEARCH THE GRAPH *)
( 10)   30              IF DNUM=PNO[I] THEN         (* IF RULE POINTS TO US...*)
( 10)   31                 IF V1<=PREM[I] THEN      (* AND IF V1 IS SUBSET OF PREMISE OF RULE *)
( 10)   32                    IF NOT INSIDE(DNUM,V2,CONS[I],TRUE) THEN(* AND IF V2 IS NOT IN THE CONSEQUENCE OF RULE *)
( 10)   33                    V1:=CONS[I];          (* THEN MAKE V1 THE CONSEQUENCE OF THE RULE *)
( 10)   34              TRSLT:=V1;
( 10)   35              FOR I:=NELE DOWNTO 1 DO     (* NOW GOING DOWN THE GRAPH *)
( 10)   36              IF DNUM=PNO[I] THEN         (* IF IT POINTS AT US ... *)
( 10)   37                 IF CONS[I]<=TRSLT THEN   (* AND THE CONSEQUENCE IS SUBSET OF US *)
( 10)   38                 TRSLT:=TRSLT+PREM[I];    (* ADD THE PREMISE *)
( 10)   39              END(*WITH*)
( 10)   40
( 10)   41           END;(*CASE STMT*)
( 10)   42
( 10)   43        FIXIT :=FIXIT*TRSLT;              (* THIS LOCUTION IS A HOLDOVER FROM THE DEC PASCAL *)
( 10)   44        END;(* EXTND *)
( 11)    1  *DECK ILINE
( 11)    2  PROCEDURE ILINE;
( 11)    3  LABEL
( 11)    4        1;
( 11)    5  BEGIN
( 11)    6        IF INFILE=0 THEN                  (* IF WE'RE READING FROM TERMINAL *)
( 11)    7  1:    BEGIN(*BREAK;*)
( 11)    8                                          (* CAUSE ACCUMULATED OUTPUT TO GO TO TTY ON DEC10 *)
( 11)    9           IF EOF(IFILE) THEN RESET(IFILE); (* IF HE HIT <CR> LAST TIME, WE MUST *)
( 11)   10           READLN(IFILE);                 (* RESET FILE TO AVOID READING BEYOND EOF *)
( 11)   11           IF EOF(IFILE) THEN GOTO 1;     (* IF HE JUST HITS <CR>, GIVE HIM ANOTHER PROMPT *)
( 11)   12           END
( 11)   13        ELSE BEGIN                        (* IF READING FROM CFILE...*)
( 11)   14           IF EOF(CFILE) THEN BEGIN
( 11)   15              INFILE:=0;                  (* WHEN WE HIT EOF, SWITCH TO READING TTY *)
( 11)   16              GOTO 1;
( 11)   17              END;
( 11)   18           READLN(CFILE);                 (* GET IT FROM CFILE *)
( 11)   19           IF EOF(CFILE) THEN BEGIN
( 11)   20              INFILE:=0;
( 11)   21              GOTO 1;
```

```
( 11)   22              END;
( 11)   23            END;
( 11)   24         END;
( 12)    1   *DECK PEOS
( 12)    2   FUNCTION PEOS
( 12)    3        (I                : INTEGER): BOOLEAN;
( 12)    4   BEGIN
( 12)    5         PEOS:=FALSE;
( 12)    6         IF INFILE=0 THEN
( 12)    7            IF EOLN(IFILE) THEN PEOS:=TRUE ELSE
( 12)    8            ELSE IF EOLN(CFILE) THEN PEOS:=TRUE;
( 12)    9         END;
( 13)    1   *DECK GETCHRR
( 13)    2   PROCEDURE GETCHRR
( 13)    3        (VAR C            : CHAR);
( 13)    4   BEGIN
( 13)    5         IF PEOS(1) THEN ILINE;       (* IF WE ARE AT END OF LNE, GET ANOTHER *)
( 13)    6         IF INFILE=0 THEN READ(IFILE,C) ELSE READ(CFILE,C);
( 13)    7         END;                                (* GETCHRR *)
( 14)    1   *DECK INIT
( 14)    2   PROCEDURE INIT;
( 14)    3   VAR
( 14)    4         I,J               : INTEGER;
( 14)    5   BEGIN
( 14)    6         RESET(IFILE);     (*  RESET TELETYPE INPUT FILE *)
( 14)    7         TRACE:=[];                          (* NO TRACES *)
( 14)    8         PRULE:=TRUE;                        (* PGRAPH SHOULD PRINT THE WHOLE RULE *)
( 14)    9         STP:=[];                            (* NO STOPS *)
( 14)   10         GSET:=NIL;                          (* NO RULES YET *)
( 14)   11         FREEG:=NIL;                         (* NO FREE GRAPH STRUCTURES YET *)
( 14)   12         FREECPXTAB:=NIL;                    (* NO FREE CPXTABLES YET *)
( 14)   13         ARITHHEAD := NIL;                   (* NO ARITH DERIVED DESCRIPTORS YET *)
( 14)   14         RESTLIST:=NIL;                      (* NO RESTRICTIONS *)
( 14)   15         DST.NELE:=0;                        (* NO DOMAIN STRUCTURE DESCRIPTIONS *)
( 14)   16         MST.NMST:=0;                        (* NO META SELECTORS *)
( 14)   17         MST.METATRIM:=3;                    (* DEFAULT: BUILD 3 METASELECTORS *)
( 14)   18         FOR I:=1 TO GSIZE DO MST.PTR[I]:=I;(* INIT PTR TO POINT TO ITSELF *)
( 14)   19         INFILE:=0;                          (* START READING FROM TTY *)
( 14)   20         RESET(TABLES);                      (* OPEN THE PARSE TABLE FILE *)
( 14)   21         FOR I:=1 TO MAXCSTF DO                   (* INITIALIZE COST FUNCTIONS *)
( 14)   22         BEGIN
( 14)   23            AQP.CSTF[I]:=I;                  (* ORDER OF APPLICATION IS 1 2 3 4 5 6 *)
( 14)   24            PRM.CSTF[I]:=I;                  (* ORDER OF APPLICATION IS 1 2 3 4 5 6 *)
( 14)   25            PRM.TOLER[I]:=0.0;               (* TOLERANCES ARE 0 *)
( 14)   26            AQP.TOLER[I]:=0;(* VL1 TOLERANCES ARE 0 *)
( 14)   27            END;
( 14)   28         PRM.CSTF[1]:=3;                     (* INIT DEFAULT COST FUNCTIONS  APPLICATION IS 3 -1 2 *)
( 14)   29         PRM.CSTF[2]:=-1;       PRM.CSTF[3]:=2;
( 14)   30         PRM.TOLER[1]:=0.3;                  (* TOLERANCES ARE 0.3 0 0 *)
( 14)   31         AQP.CSTF[1]:=-1;                    (* INIT VL1 COST FUNCTIONS  -1 2 4 3 *)
( 14)   32         AQP.CSTF[5]:=-5;                    (* -5 *)
( 14)   33         AQP.CSTF[3]:=4;        AQP.CSTF[4]:=3;
( 14)   34         AQP.NF:=2;                          (* ONLY USE THE FIRST 2 VL1 COST FUNCTIONS -1 2 *)
( 14)   35         PRM.NF:=3;                          (* ONLY USE THE FIRST 3 VL21 COST FUNCTIONS 3 -1 2 *)
( 14)   36         PRM.NCONSIST:=4;                    (* BUILD 4 CONSISTENT GENERALIZATIONS BEFORE AQ *)
( 14)   37         STAR:=NIL;                          (* NO STAR *)
( 14)   38         PSTAR:=NIL;                         (* NO PARTIAL STAR *)
( 14)   39         COVSET:=NIL;                        (* NO COVERING SET (F1) *)
```

```
( 14)   40        PRM.MAXSTAR:=2;                    (* TRIM VL21 STARS TO 2 RULES *)
( 14)   41        AQP.MAXSTARAQ:=2;                  (* TRIM VL1 STARS TO 2 RULES *)
( 14)   42        AQP.LQST:=TRUE;                    (* PRUNE THE REFERENCE AFTER VL1 PROCEDURE *)
( 14)   43        PRM.EXTMTY:=FALSE;                 (* NO MST- OR LST- SELECTORS YET *)
( 14)   44        PRM.EQUIV:=FALSE;                  (* NO =SAME SELECTORS YET *)
( 14)   45        AQP.CUTF1:=20;                     (* NO MORE THAN 20 ELEMENTS IN CUTSTAR *)
( 14)   46        NEWTRACE:=0;                       (* NO TRACING IN VL21 MODE *)
( 14)   47        AQP.FREEC:=NIL;                    (* NO FREE COMPLEXES YET *)
( 14)   48        AQP.NVAR:=0;                       (* ?? *)
( 14)   49        CRULENO:=1;                        (* CURRENT RULE NUMBER IS 1 *)
( 14)   50        PRM.ALTER:=2;                      (* GENERATE 2 ALTERNATIVES FROM EVERY RULE IN THE STAR IN VL21 *)
( 14)   51        PRM.DESCTYPE := DISCRIMINANT;      (* PRODUCE DISCRIMINANT COVER BY DEFAULT *)
( 14)   52        PRM.MINCOVER := 10;                (* DEFAULT MIN COVER FOR CHARACTERISTIC IS 10 %*)
( 14)   53        FOR I:= 1 TO MAXPTSIZE DO                  (* GET THE PARSE TABLE *)
( 14)   54        BEGIN
( 14)   55           READLN(TABLES);      READ(TABLES,J);(* GET THE CONTINUATION FLAG *)
( 14)   56           IF J=1 THEN PTBL.CONT[I]:=TRUE(* IF 1, THEN THIS ROW IS CONT OF PREVIOUS ROW *)
( 14)   57           ELSE PTBL.CONT[I]:=FALSE;
( 14)   58           READ(TABLES,PTBL.SRULE[I]);    (* GET THE SEMANTIC RULE *)
( 14)   59           J:=1;
( 14)   60           REPEAT
( 14)   61              READ(TABLES,CHRR);          (* GET THE RIGHT HAND SIDE OF THE RULE *)
( 14)   62              IF CHRR<>' ' THEN           (* NUMERIC FIELDS ARE ALL PRECEDED BY A BLANK *)
( 14)   63              PTBL.RHS[I,J]:=ORD(CHRR)    (* SO A NON-BLANK IS A TERMINAL *)
( 14)   64              ELSE READ(TABLES,PTBL.RHS[I,J]);(* GET THE NON-TERMINAL *)
( 14)   65              J:=J+1;
( 14)   66              UNTIL PTBL.RHS[I,J-1]=0;
( 14)   67           FOR J:=1 TO 60 DO PTBL.NAME[I,J]:=CHR(0); (* INIT THE PRODUCTION NAME TO NULLS *)
( 14)   68                 (* INITIALIZE THE NAME *)
( 14)   69           J:= 1;         (*   READ IN THE NAME FOR THIS PRODUCTION *)
( 14)   70           WHILE (NOT EOLN(TABLES)) AND (J<=60) DO BEGIN
( 14)   71              READ(TABLES,PTBL.NAME[I,J]);       J:=J+1;
( 14)   72              END;     (*  WHILE  NOT EOLN *)
( 14)   73           END;(*FOR I:= *)
( 14)   74
( 14)   75        RESET(STAB);                       (* OPEN THE STAB FILE AND READ IN OLD SYMBOL TABLE *)
( 14)   76        S.NELT:=0;
( 14)   77        FOR I:=1 TO SYMSZE DO BEGIN
( 14)   78           S.NVAL[I]:=0;                   (* NO VALUES FOR THIS NODE *)
( 14)   79           S.VTYPE[I]:=1;                  (* DEFAULT IS NOMINAL DOMAIN *)
( 14)   80           S.EVAL[I]:=0;                   (* NO VALUES... *)
( 14)   81           S.MVAL[I]:=MNVAL;               (* INITIAL SET OF VALUES IS EVERYTHING *)
( 14)   82           S.DPNO[I] := 0;                      S.PNO[I] := 0;
( 14)   83           FOR J:=1 TO 10 DO S.NAME[I,J]:=' ';
( 14)   84           END;
( 14)   85        FOR I:=-SYMSZE TO SYMSZE DO BEGIN
( 14)   86           S.NARG[I]:=0;       S.VCOST[I]:=0
( 14)   87           END;
( 14)   88        S.NAME[S.NELT+1]:='FORALL     ';  (* ADD THE FORALL AND #PT ENTRIES INTO S *)
( 14)   89        S.NAME[S.NELT+2]:='#PT        ';
( 14)   90        S.NELT:=S.NELT+2;      S.PNO[S.NELT-1]:=S.NELT-1;
( 14)   91        S.PNO[S.NELT]:=S.NELT;             (* ALL META SELECTORS POINT TO THESE*)
( 14)   92        S.VTYPE[S.NELT]:=2;                (* TWO ENTRIES .. *)
( 14)   93        S.MVAL[S.NELT]:=0;     S.NVAL[S.NELT]:=0;
( 14)   94        S.EVAL[S.NELT]:=0;     S.MVAL[S.NELT-1]:=0;
( 14)   95        S.NVAL[S.NELT-1]:=1;   S.EVAL[S.NELT-1]:=1;
( 14)   96        IF NOT EOF(STAB) THEN              (* TRY TO READ SYMBOL TABLE FROM STAB *)
( 14)   97        S:=STAB^;       RESET(DFILE);      (* TRY TO READ DOMAIN DEFNS FROM DFILE *)
```

```
( 14)   98        IF NOT EOF(DFILE) THEN DST:=DFILE^;
( 14)   99        END; (* INIT *)
( 15)    1   *DECK NEWG
( 15)    2   PROCEDURE NEWG
( 15)    3        (VAR G           : GPTR);(*     PURPOSE:
( 15)    4          TO OBTAIN A NEW GRAPH STRUCTURE, EITHER FROM THE FREEG LIST
( 15)    5        OR VIA A CALL TO NEW.
( 15)    6    *)
( 15)    7   BEGIN
( 15)    8        G:=FREEG;
( 15)    9        IF FREEG=NIL THEN NEW(G) ELSE FREEG:=G^.NXTN;
( 15)   10        G^.FP:=TRUE;                          (* INITIALIZE THE GRAPH.  DENOTE IT AS ACTIVE *)
( 15)   11        G^.RNO:=CRULENO;                      (* ASSIGN IT THE NEXT RULE NUMBER *)
( 15)   12        G^.MSEL:=NIL;                         (* NO METASELECTOR CPX ATTACHED YET *)
( 15)   13        G^.NNEG := NIL;
( 15)   14        FOR I:=1 TO 5 DO G^.COST[I] := 0;
( 15)   15        FOR I:=1 TO GSIZE DO G^.COUNT[I] := 0;
( 15)   16        CRULENO:=CRULENO+1;                   (* INCREMENT RULE NUMBER *)
( 15)   17        END;(* NEWG *)
( 16)    1   *DECK GIN
( 16)    2   PROCEDURE GIN
( 16)    3        (G               : GPTR);
( 16)    4   BEGIN
( 16)    5        G^:=GFILE^;       GET(GFILE);
( 16)    6        END;
( 17)    1   *DECK GOUT
( 17)    2   PROCEDURE GOUT
( 17)    3        (G               : GPTR);
( 17)    4   BEGIN
( 17)    5        GFILE^:=G^;       PUT(GFILE);
( 17)    6        END; (*  GOUT *)
( 18)    1   *DECK NEWCPXTAB
( 18)    2   PROCEDURE NEWCPXTAB
( 18)    3        (VAR T           : CPXTABPTR);(* PURPOSE:  TO ALLOCATE A NEW CPXTAB TO T *)
( 18)    4   BEGIN
( 18)    5        IF FREECPXTAB=NIL THEN BEGIN     (* IF NONE ON FREE LIST, GET A NEW ONE *)
( 18)    6           NEW(FREECPXTAB);       FREECPXTAB^.NXT:=NIL;(* AND TERMINATE THE LIST WITH NIL *)
( 18)    7           END;(* IF NIL *)
( 18)    8        T:=FREECPXTAB;                        (* ASSIGN THE TABLE TO THIS GUY *)
( 18)    9        FREECPXTAB:=FREECPXTAB^.NXT;          (* AND ADVANCE THE HEAD OF LIST POINTER *)
( 18)   10        T^.COVCOUNT:=0;                       (* INITIALIZE COVCOUNT AND NXT *)
( 18)   11        T^.NXT:=NIL;
( 18)   12        END;(*NEXCPXTAB*)
( 19)    1   *DECK FREETAB
( 19)    2   PROCEDURE FREETAB
( 19)    3        (VAR T           : CPXTABPTR);(* PURPOSE:  TO PUT CPXTABLE T ONTO THE FREE CPXTABLE LIST *)
( 19)    4   BEGIN
( 19)    5        IF T<>NIL THEN BEGIN
( 19)    6           T^.NXT:=FREECPXTAB;                (* POINT IT AT HEAD OF LIST *)
( 19)    7           FREECPXTAB:= T;                    (* POINT HEAD OF LIST AT IT *)
( 19)    8           T:=NIL;                            (* RETURN A CLEANED POINTER *)
( 19)    9           END;(* IF T<>NIL *)
( 19)   10        END;(* FREETAB *)
( 20)    1   *DECK NEWC
( 20)    2   PROCEDURE NEWC
( 20)    3        (VAR C           : CPTR);(* PURPOSE:  TO ALLOCATE A NEW COMPLEX TO C *)
( 20)    4   BEGIN
( 20)    5        IF AQP.FREEC=NIL THEN BEGIN
```

```
( 20)    6          NEW(AQP.FREEC);                  (* IF NONE ON FREE LIST, GET A NEW ONE *)
( 20)    7          AQP.FREEC^.TAB:=NIL;             (* NO TABLE ASSOCIATED *)
( 20)    8          AQP.FREEC^.NXTC := NIL;          (* TERMINATE LIST *)
( 20)    9          END;
( 20)   10       C:=AQP.FREEC;                       (* POINT C TO HEAD ELEMENT IN LIST *)
( 20)   11       AQP.FREEC:=AQP.FREEC^.NXTC;         (* AND ADVANCE HEAD OF LIST POINTER *)
( 20)   12       FREETAB(C^.TAB);                    (* IF THERE IS AN ASSOCIATED CPXTABLE, FREE IT *)
( 20)   13       C^.NXTC:=NIL;                       (* CLEAR FORWARD POINTER *)
( 20)   14       END;(*NEWC*)
( 21)    1  *DECK FREECPX
( 21)    2  PROCEDURE FREECPX
( 21)    3       (VAR C          : CPTR);(* PURPOSE: TO FREE A LIST OF COMPLEXES.  C POINTS TO FIRST ELEMENT IN LIST
( 21)    4      SEARCH AND FIND THE END OF THE LIST AND THEN PLACE ENTIRE LIST ON
( 21)    5      THE AQP.FREEC CHAIN.  LIST TERMINATES WHEN C^.NXTC IS NIL *)
( 21)    6  VAR
( 21)    7       C1              : CPTR;
( 21)    8  BEGIN
( 21)    9       IF C<>NIL THEN BEGIN
( 21)   10          C1:=C;                           (* START AT HEAD OF LIST *)
( 21)   11          FREETAB(C1^.TAB);                    (* FREE ANY ASSOCIATED TABLE *)
( 21)   12          WHILE C1^.NXTC<>NIL DO BEGIN
( 21)   13             C1:=C1^.NXTC;      FREETAB(C1^.TAB);(* FREE THE ASSOCIATED TABLES OF EVERY CPX *)
( 21)   14             END;(* WHILE <> NIL *)
( 21)   15
( 21)   16          C1^.NXTC:=AQP.FREEC;             (* PLACE LIST ON FREE LIST *)
( 21)   17          AQP.FREEC:=C;          C:=NIL;   (* RETURN AN EMPTY LIST *)
( 21)   18          END;(* IF *)
( 21)   19       END;(* FREECPX *)
( 22)    1  *DECK EXPLN
( 22)    2  PROCEDURE EXPLN
( 22)    3       (I              : INTEGER);
( 22)    4  LABEL
( 22)    5       11;
( 22)    6  VAR
( 22)    7       CHRR            : CHAR;
( 22)    8
( 22)    9
( 22)   10    (*            I N D U C E - 1  --  EXPLN -- RDEX        *)
( 22)   11
( 22)   12
( 22)   13  *CALL RDEX
( 22)   14      (* RDEX*)
( 22)   15   (*1
( 22)   16                 I N D U C E - 1  --  EXPLN (MAIN CODE)
( 22)   17   *)
( 22)   18
( 22)   19  BEGIN
( 22)   20       IF (1<=I)AND(I<=10) THEN            (* IF I IS LEGAL *)
( 22)   21          IF I IN STP THEN                 (* AND I IS A TURNED ON STOP *)
( 22)   22          BEGIN
( 22)   23             WRITELN(OUTPUT);      WRITELN(OUTPUT, 'STOP AT TRACE LEVEL',I:2);
( 22)   24  11:        WRITELN(OUTPUT,'ENTER ? FOR EXPLANATION', ' P TO CHANGE', ' PARAMETERS OR RETURN TO CONTINUE');
( 22)   25             WRITELN(OUTPUT);
( 22)   26             IF EOF(IFILE) THEN RESET(IFILE);
( 22)   27             READLN(IFILE);
( 22)   28             IF NOT EOLN(IFILE) THEN    (* IF HE DIDN'T HIT <CR>, SEE WHAT HE HIT *)
( 22)   29             BEGIN
( 22)   30                READ(IFILE,CHRR);
```

```
( 22)   31                IF CHRR='P' THEN          (* HE WANTS TO ENTER PARAMETERS? *)
( 22)   32                ENTERP                    (* GO TO ENTER PARAMETER PROC. *)
( 22)   33                ELSE BEGIN
( 22)   34                   RDEX(I);               (* PRINT THE APPROPRIATE EXPLANATORY INFO *)
( 22)   35                   GOTO 11;
( 22)   36                   END;
( 22)   37                END;  (* IF NOT <CR> *)
( 22)   38             END   (* IF IT IS A STOP *)
( 22)   39          ELSE
( 22)   40       ELSE RDEX(I);                      (* IF IT CAN'T BE A LEGAL STOP, PRINT THE TEXT *)
( 22)   41       END;  (* EXPLN *)
( 23)    1  *DECK RDEX
( 23)    2  PROCEDURE RDEX
( 23)    3       (I               : INTEGER);
( 23)    4  LABEL
( 23)    5       99;
( 23)    6  VAR
( 23)    7       J                : INTEGER;
( 23)    8  BEGIN
( 23)    9       RESET(EXPLAIN);                    (* REWIND THE FILE OF INFO *)
( 23)   10       CHRR:=' ';                         (* ACTIVE CHARACTER *)
( 23)   11       J:=-1;                             (* MOST RECENT "EXPLAIN ENTRY" READ *)
( 23)   12       WHILE J<>I DO                      (* AS LONG AS WE HAVEN'T HIT THE ONE WE WANT *)
( 23)   13       BEGIN                              (* SKIP THRU THE FILE *)
( 23)   14          WHILE CHRR<>'!' DO              (* AS LONG AS ! IS NOT FIRST CHAR OF LINE *)
( 23)   15          BEGIN
( 23)   16             IF NOT EOF(EXPLAIN) THEN READLN(EXPLAIN);    (* READ A LINE *)
( 23)   17             IF EOF(EXPLAIN) THEN        (* IF EOF, WE COULN'T FIND THAT NUMBER *)
( 23)   18             BEGIN
( 23)   19                WRITELN(OUTPUT,'NO HELP');
( 23)   20                GOTO 99;
( 23)   21                END;
( 23)   22             READ(EXPLAIN,CHRR);         (* READ FIRST CHAR ON THE NEW LINE *)
( 23)   23             END;
( 23)   24          READ(EXPLAIN,J);               (* READ THE NUMBER WHICH FOLLOWS THE ! *)
( 23)   25          CHRR:=' ';                     (* THAT NUMBER IS THE "EXPLAIN ENTRY NUMBER" *)
( 23)   26          END;
( 23)   27       WRITELN(OUTPUT);                   (* WE FOUND IT,  PRINT IT *)
( 23)   28       WRITELN(OUTPUT);
( 23)   29       REPEAT
( 23)   30          READLN(EXPLAIN);                (* GET A NEW LINE *)
( 23)   31          WRITELN(OUTPUT);                (* OUTPUT THE LAST LINE *)
( 23)   32          WHILE NOT EOLN(EXPLAIN) DO BEGIN
( 23)   33             READ(EXPLAIN,CHRR);          (* COPY THE LINE TO THE TTY *)
( 23)   34             WRITE(OUTPUT,CHRR);
( 23)   35             END;
( 23)   36          IF CHRR='*' THEN               (* IF A LINE ENDS IN AN ASTERISK *)
( 23)   37          BEGIN
( 23)   38             WRITELN(OUTPUT);            (* ASK USER TO HIT <CR> ALLOW HIM TIME TO READ *)
( 23)   39             WRITE(OUTPUT,'PRESS RETURN TO CONTINUE');
( 23)   40             WRITELN(OUTPUT);(*BREAK;*)
( 23)   41                                         (* SEND OUTPUT TO TTY ON DEC 10 *)
( 23)   42             IF EOF(IFILE) THEN RESET(IFILE);
( 23)   43             READLN(IFILE);
( 23)   44             END;
( 23)   45          UNTIL CHRR='!';                (* CONTINUE COPYING UNTIL LAST CHAR OF LINE IS ! *)
( 23)   46       WRITELN(OUTPUT);                  (* SKIP A FEW LINES AND EXIT *)
( 23)   47       WRITELN(OUTPUT);            WRITELN(OUTPUT);
```

```
( 23)   48  99:
( 23)   49        END;
( 24)    1  *DECK PMETAD
( 24)    2  PROCEDURE PMETAD;
( 24)    3  VAR
( 24)    4        I,TEMP1          : INTEGER;
( 24)    5  BEGIN                                  (* PRINT METADESCRIPTOR TABLE *)
( 24)    6        WRITELN(OUTPUT,'THE SELECTED META-SELECTORS ARE:');
( 24)    7        WRITELN(OUTPUT,' MS TYPE          FUNCTION       F1COV FOCOV');
( 24)    8        FOR I:=1 TO MST.NMST DO BEGIN
( 24)    9           WRITE(OUTPUT,I:3,'   ');       TEMP1 := MST.PTR[I];       WRITE(OUTPUT,S.NAME[MST.SYMPTR[TEMP1]]);
( 24)   10           WRITE(OUTPUT,S.NAME[MST.PNO[TEMP1]]);                     WRITE(OUTPUT,'=',MST.VAL[TEMP1]:3);
( 24)   11           WRITE(OUTPUT,'  ',MST.F1COV[TEMP1]:5);                    WRITE(OUTPUT,',',MST.FOCOV[TEMP1]:5);
( 24)   12           WRITELN(OUTPUT);
( 24)   13           END;
( 24)   14        END; (* PMETAD *)
( 25)    1  *DECK ENTERP
( 25)    2  PROCEDURE ENTERP;(*     PURPOSE:  THIS PROCEDURE ALLOWS THE USER TO INSPECT THE RULE
( 25)    3       BASE AND THE PARAMETERS AND CHANGE THE PARAMETERS OF THE
( 25)    4       PROGRAM AT ANY TIME.
( 25)    5   *)
( 25)    6  LABEL
( 25)    7        1,
( 25)    8        2,
( 25)    9        3,
( 25)   10        4,
( 25)   11        5,
( 25)   12        6,
( 25)   13        7,
( 25)   14        8,
( 25)   15        9;
( 25)   16  TYPE
( 25)   17        NTYPE            = PACKED ARRAY[1..11] OF CHAR;
( 25)   18  VAR
( 25)   19        NAME             : ARRAY[1..28] OF NTYPE;
( 25)   20        BUF              : ARRAY[0..80] OF CHAR;
( 25)   21        I,J,K1,K,L,M,BLEN,ARG: INTEGER;      (* BLEN IS BUFFER LENGTH *)
( 25)   22        LCHAR            : CHAR;       (*  TEMP FOR SKIPPING LEADING BLANKS *)
( 25)   23        G                : GPTR;(*
( 25)   24                  I N D U C E - 1  --  ENTERP -- PDOM
( 25)   25
( 25)   26   *)
( 25)   27  *CALL PDOM
( 25)   28      (*PDOM*)
( 25)   29   (*
( 25)   30                  I N D U C E - 1  --  ENTERP -- PRINTPS
( 25)   31
( 25)   32   *)
( 25)   33
( 25)   34  *CALL PRINTPS
( 25)   35      (*PRINTPS*)
( 25)   36   (*
( 25)   37                  I N D U C E - 1  --  ENTERP -- GETNUM
( 25)   38
( 25)   39   *)
( 25)   40  *CALL GETNUM
( 25)   41      (*GETNUM*)
( 25)   42   (*1
```

```
( 25)   43                I N D U C E - 1  --  ENTERP (MAIN CODE)
( 25)   44    *)
( 25)   45
( 25)   46  BEGIN                                   (* INIT THE NAMES OF THE PARAMETERS *)
( 25)   47        NAME[1]:='TRACE       ';      NAME[2]:='AQCUTF1     ';      NAME[3]:='AQMAXSTAR  ';      NAME[4]:='AQTOLERANCE';
( 25)   48        NAME[5]:='AQCRIT      ';      NAME[6]:='AQNF        ';      NAME[7]:='VCOST      ';      NAME[8]:='VLMAXSTAR  ';
( 25)   49        NAME[9]:='VLTOLERANCE';      NAME[10]:='VLCRIT      ';      NAME[11]:='VLNF       ';      NAME[12]:='NCONSIST   ';
( 25)   50        NAME[13]:='ALTER      ';      NAME[14]:='PRINT       ';      NAME[15]:='METATRIM   ';      NAME[16]:='DESCTYPE   ';
( 25)   51        NAME[17]:='QUIT       ';      NAME[18]:='HELP PARAM  ';      NAME[19]:='PARAMETERS ';      NAME[20]:='STP        ';
( 25)   52        NAME[21]:='MINCOVER   ';      NAME[22]:='QUICK       ';      NAME[23]:='DETAIL     ';      NAME[24]:='EXPLAIN    ';
( 25)   53        NAME[25]:='BRIEF      ';      NAME[26]:='VTYPE       ';      NAME[27]:='PRULE      ';      NAME[28]:='LQST       ';
( 25)   54  2:    IF INFILE=0 THEN WRITELN(OUTPUT,'ENTER RULE  # TO SEE RULE', ' PARA OR PARM', '=VALUE TO SEE OR CNG.PARM',
( 25)   55       ' HELP OR QUIT');
( 25)   56        ILINE;
( 25)   57        FOR BLEN:=0 TO 80 DO BUF[BLEN]:=' ';
( 25)   58        REPEAT
( 25)   59           GETCHRR(LCHAR);          (*  SKIP LEADING BLANKS *)
( 25)   60           UNTIL LCHAR<>' ';
( 25)   61        BUF[1]:=LCHAR;       BLEN:=1;
( 25)   62        WHILE NOT PEOS(I) DO BEGIN
( 25)   63           BLEN:=BLEN+1;       GETCHRR(BUF[BLEN]);
( 25)   64           END;
( 25)   65        J:=0;       GETNUM(J,I);          (* GET 2 NUMBERS FROM THE LINE *)
( 25)   66        GETNUM(J,L);
( 25)   67        IF BUF[1] IN ['0'..'9'] THEN     (* IF THE FIRST CHAR IS A NUMBER, HE WANTS THE RULE PRINTED *)
( 25)   68        BEGIN                            (* SEARCH THE VARIOUS RULE LISTS TO TRY TO FIND HIS RULE *)
( 25)   69           G:=GSET;
( 25)   70           WHILE G<>NIL DO IF G^.RNO=I THEN GOTO 1 ELSE G:=G^.NXTN;
( 25)   71           G:=STAR;
( 25)   72           WHILE G<>NIL DO IF G^.RNO=I THEN GOTO 1 ELSE G:=G^.NXTN;
( 25)   73           G:=PSTAR;
( 25)   74           WHILE G<>NIL DO IF G^.RNO=I THEN GOTO 1 ELSE G:=G^.NXTN;
( 25)   75           G:=COVSET;
( 25)   76           WHILE G<>NIL DO IF G^.RNO=I THEN GOTO 1 ELSE G:=G^.NXTN;
( 25)   77           G:=RESTLIST;
( 25)   78           WHILE G<>NIL DO IF G^.RNO=I THEN GOTO 1 ELSE G:=G^.NXTN;
( 25)   79           WRITELN(OUTPUT,'RULE',I,' NOT FOUND');
( 25)   80           GOTO 2;
( 25)   81  1:       PGRAPH(G,S);                 (* PRINT THE RULE *)
( 25)   82           GOTO 2;
( 25)   83           END(*IF BUF IN*)
( 25)   84
( 25)   85        ELSE FOR K:=1 TO 28 DO           (* ELSE SEARCH 1ST 4 CHARS FOR A VALID COMMAND *)
( 25)   86        BEGIN
( 25)   87           FOR K1:=1 TO 4 DO IF BUF[K1]<>NAME[K,K1] THEN GOTO 3;
( 25)   88           GOTO 5;
( 25)   89  3:       ;
( 25)   90           END;
( 25)   91        WRITELN(OUTPUT,'TRY AGAIN');
( 25)   92        GOTO 2;
( 25)   93  5:.   ARG := 1;                        (* COMPUTE INDEX OF ARGUMENT, IF ANY *)
( 25)   94        WHILE (BUF[ARG]<>' ') AND (ARG<BLEN) DO ARG:=ARG+1; (* SKIP COMMAND *)
( 25)   95        WHILE (BUF[ARG]=' ') AND (ARG<BLEN) DO ARG:=ARG+1; (* SKIP BLANKS *)
( 25)   96        IF (BUF[ARG]=' ') THEN ARG := 0;    (* IF WE HIT BLEN, INDICATE BAD ARG *)
( 25)   97        CASE K OF                        (* CASE THE SUBSCRIPT OF THE COMMAND TABLE -- PROCESS COMMAND *)
( 25)   98  1:       IF I<0 THEN TRACE:=TRACE-[ABS(I)] ELSE TRACE:=TRACE+[ABS(I)];
( 25)   99  2:       AQP.CUTF1:=I;
( 25)  100  3:       AQP.MAXSTARAQ:=I;
```

```
( 25)  101  4:       AQP.TOLER[I]:=L/100.0;
( 25)  102  5:       IF I>0 THEN AQP.CSTF[I]:=L;
( 25)  103  6:       AQP.NF:=I;
( 25)  104  14:      BEGIN
  25)  105             IF BUF[ARG] IN ['M','R','D','F'] THEN BEGIN
( 25)  106               CASE BUF[ARG] OF
( 25)  107                 'M': PMETAD;
( 25)  108                 'R': G:=RESTLIST;
( 25)  109                 'D': PDOM;
( 25)  110                 'F': G:=GSET
( 25)  111                 END;(*CASE STMT*)
( 25)  112
( 25)  113               IF BUF[ARG] IN ['R','F'] THEN WHILE G<>NIL DO BEGIN
( 25)  114                 PGRAPH(G,S);      G:=G^.NXTN;
( 25)  115                 END;
( 25)  116               END(*IF*)
( 25)  117
( 25)  118             ELSE BEGIN
( 25)  119               WRITELN(OUTPUT,'ENTER PRINT X WHERE X IS');      WRITELN(OUTPUT,'     (M) PRINT META DESCRIPTORS');
( 25)  120               WRITELN(OUTPUT,'     (F) PRINT INPUT DECISION RULES');
( 25)  121               WRITELN(OUTPUT,'     (D) DOMAIN INFORMATION');   WRITELN(OUTPUT,'     (R) RESTRICTIONS');
( 25)  122               END;
( 25)  123             END;
( 25)  124  15:      MST.METATRIM:=I;
( 25)  125  7,26:    BEGIN
( 25)  126             L:=0;       M:=0;
( 25)  127             FOR J:=1 TO BLEN DO IF BUF[J]='(' THEN L:=J+1
( 25)  128             ELSE IF BUF[J]=')' THEN M:=J-1;
( 25)  129             IF M*L = 0 THEN BEGIN
( 25)  130               WRITELN(OUTPUT,'INVALID SYNTAX');
( 25)  131               GOTO 2;
( 25)  132               END;
( 25)  133             FOR J:=1 TO S.NELT DO BEGIN
( 25)  134               FOR K1:=L TO M DO IF BUF[K1]<>S.NAME[J,K1-L+1] THEN GOTO 8;
( 25)  135               GOTO 9;
( 25)  136  8:           ;
( 25)  137               END;
( 25)  138             WRITELN(OUTPUT,'DESCRIPTOR NOT FOUND IN STAB');
( 25)  139             GOTO 2;
( 25)  140  9:         IF K=7 THEN S.VCOST[J]:=I ELSE S.VTYPE[J]:=I;
( 25)  141             GOTO 2;
( 25)  142             END;(*CASE 7*)
( 25)  143
( 25)  144  8:       PRM.MAXSTAR:=I;
( 25)  145  9:       PRM.TOLER[I]:=L/100.0;
( 25)  146  10:      IF I>0 THEN PRM.CSTF[I]:=L;
( 25)  147  11:      PRM.NF:=I;
( 25)  148  12:      PRM.NCONSIST:=I;
( 25)  149  13:      PRM.ALTER:=I;
( 25)  150  27:      IF (BUF[ARG]='F') AND (ARG>0) THEN PRULE:=FALSE ELSE PRULE:=TRUE;
( 25)  151  28:      IF (BUF[ARG]='F') AND (ARG>0) THEN AQP.LQST:=FALSE ELSE AQP.LQST:=TRUE;
( 25)  152  16:      IF (ARG>0) AND (BUF[ARG]='C') THEN PRM.DESCTYPE := CHARACTERISTIC ELSE PRM.DESCTYPE := DISCRIMINANT;
( 25)  153                                          (* SET DESCRIPTION TYPE *)
( 25)  154  21:      PRM.MINCOVER := I;             (* GET MIN COVER VALUE *)
( 25)  155  17:      GOTO 4;
( 25)  156  18:      BEGIN
( 25)  157             FOR I:=1 TO 28 DO BEGIN
( 25)  158               FOR K1:=1 TO 4 DO IF BUF[K1+5]<>NAME[I,K1] THEN GOTO 6;
```

```
 25)  159                EXPLN(100*I);
 25)  160                GOTO 7;
 25)  161  6:            ;
 25)  162                END;
 25)  163             EXPLN(18);      WRITELN(OUTPUT,' THE VALID PARAMETERS ARE:');
 25)  164             FOR I:=1 TO 28 DO WRITELN(OUTPUT,NAME[I]);
 25)  165  7:         ;
 25)  166             END;(*CASE 18*)
 25)  167
 25)  168  19:     PRINTPS;
 25)  169  20:     IF I<0 THEN STP:=STP-[ABS(I)] ELSE STP:=STP+[ABS(I)];
 25)  170  22:     TRACE:=[];
 25)  171  23:     BEGIN
 25)  172             TRACE:=[1..10];      STP:=[];
 25)  173             END;
 25)  174  24:     BEGIN
 25)  175             TRACE:=[1..10];      STP:=[1..10];
 25)  176             END;
 25)  177  25:     BEGIN
 25)  178             TRACE:=[3,9,10];      STP:=[10];
 25)  179             END
 25)  180          END;(*CASE STMT*)
 25)  181
 25)  182     GOTO 2;
 25)  183  4: ;
 25)  184     END;(*ENTERP*)
 26)    1  *DECK PDOM
 26)    2  PROCEDURE PDOM;                          (* PRINT DOMAIN INFORMATION *)
 26)    3  VAR
 26)    4     I,J,K          : INTEGER;
 26)    5  BEGIN
 26)    6     WRITELN(OUTPUT,'   NAME    ',' NARG ',' TYPE ', ' COST ',' MIN ',' MAX ', '  STRUCTURE');
 26)    7     WITH S DO FOR I:=1 TO NELT DO BEGIN
 26)    8        WRITE(OUTPUT,NAME[I],NARG[I]:4,VTYPE [I]:6,VCOST[I]:6,MVAL[I]:5,EVAL[I]:5);
 26)    9        IF VTYPE[I]=3 THEN FOR J:=1 TO DST.NELE DO
 26)   10           IF I=DST.PNO[J] THEN BEGIN
 26)   11              FOR K:=0 TO MNVAL DO IF K IN DST.PREM[J] THEN WRITE(OUTPUT,K:2);
 26)   12              WRITE(OUTPUT,'=>');
 26)   13              FOR K:=0 TO MNVAL DO IF K IN DST.CONS[J] THEN WRITE(OUTPUT,K:2);
 26)   14              WRITE(OUTPUT,';  ');
 26)   15              END;
 26)   16        WRITELN(OUTPUT);
 26)   17        END;
 26)   18     END;
 27)    1  *DECK PRINTPS
 27)    2  PROCEDURE PRINTPS;                       (* PRINT THE PARAMETERS *)
 27)    3  VAR
 27)    4     I,J            : INTEGER;
 27)    5
 27)    6   (*1
 27)    7              I N D U C E - 1  --  ENTERP -- PRINTPS -- WTOLER
 27)    8   *)
 27)    9  *CALL WTOLER
 27)   10   (*              I N D U C E - 1  --  ENTERP  --  PRINTPS  --  MAX
 27)   11   *)
 27)   12  *CALL MAX
 27)   13
 27)   14   (*              I N D U C E - 1  --  ENTERP -- PRINTPS(MAIN CODE)
```

```
( 27)   15   *)
( 27)   16
( 27)   17  BEGIN
( 27)   18        WRITELN(OUTPUT);       WRITELN(OUTPUT);       WRITE(OUTPUT,'           TRACE=');
( 27)   19        FOR I:=0 TO 10 DO IF I IN TRACE THEN WRITE(OUTPUT,I:3);
( 27)   20        WRITELN(OUTPUT);       WRITE(OUTPUT,'           STOPS=');
( 27)   21        FOR I:=0 TO 10 DO IF I IN STP THEN WRITE(OUTPUT,I:3);
( 27)   22        WRITELN(OUTPUT);
( 27)   23        IF PRULE THEN WRITE(OUTPUT,'          PRINT RULES AND RULE NUMBERS');
( 27)   24        WRITELN(OUTPUT);       WRITELN(OUTPUT);
( 27)   25        WRITELN(OUTPUT,'             VARIABLE NAME ', '                  VTYPE', '   VCOST');
( 27)   26        FOR I:=1 TO S.NELT DO IF (S.VCOST[I]<>0) OR (S.VTYPE[I]<>1) THEN BEGIN
( 27)   27           WRITE(OUTPUT,'                    ');
( 27)   28           FOR J:=1 TO 10 DO WRITE(OUTPUT,S.NAME[I,J]);
( 27)   29           WRITE(OUTPUT,'          ');       WRITELN(OUTPUT,'        ',S.VTYPE[I]:1,S.VCOST[I]:9);
( 27)   30           END;
( 27)   31        WRITELN(OUTPUT);       WRITELN(OUTPUT, '              AQPARMS', '              ', '                    VLPARMS');
( 27)   32        WRITE(OUTPUT,'      ',' AQMAXSTAR = ', AQP.MAXSTARAQ:3);
( 27)   33        WRITE(OUTPUT,'     ');
( 27)   34        IF AQP.LQST THEN WRITE(OUTPUT,' LQST') ELSE WRITE(OUTPUT,'     ');
( 27)   35        WRITE(OUTPUT,'   NCONSIST = ', PRM.NCONSIST:3, ' ALTER = ', PRM.ALTER:3);
( 27)   36        WRITELN(OUTPUT, ' VLMAXSTAR = ',PRM.MAXSTAR:3);
( 27)   37        WRITELN(OUTPUT);
( 27)   38        WRITELN(OUTPUT, '       AQCRIT       ',' AQTOLERANCE          VLCRIT', '         VLTOLERANCE');
( 27)   39        FOR I:=1 TO MAX(AQP.NF,PRM.NF) DO BEGIN
( 27)   40           IF I<=AQP.NF THEN BEGIN
( 27)   41              WRITE(OUTPUT,'       ',AQP.CSTF[I]:2,'              ');       WTOLER(AQP.TOLER[I]);
( 27)   42              END
( 27)   43           ELSE WRITE(OUTPUT,'                                    ');
( 27)   44
( 27)   45           IF I<=PRM.NF THEN BEGIN
( 27)   46              WRITE(OUTPUT,'                ',PRM.CSTF[I]:2,'                ');       WTOLER(PRM.TOLER[I]);
( 27)   47              END;(*IF I<=PRM.NF*)
( 27)   48           WRITELN(OUTPUT);
( 27)   49           END;(* FOR I *)
( 27)   50        WRITELN(OUTPUT);
( 27)   51        WRITELN(OUTPUT, 'NUMBER OF COST CRITERIA:  AQNF = ', AQP.NF:3, '              ' , '            VLNF = ', PRM.NF:3);
( 27)   52        WRITE(OUTPUT,'NEW FUNCTIONS:       METATRIM = ',MST.METATRIM:3);
( 27)   53        IF PRM.EXTMTY THEN WRITE(OUTPUT,'    EXTMTY');
( 27)   54        IF PRM.EQUIV THEN WRITE(OUTPUT,'    EQUIV');
( 27)   55        WRITELN(OUTPUT);
( 27)   56        CASE PRM.DESCTYPE OF
( 27)   57  DISCRIMINANT:
( 27)   58           WRITELN(OUTPUT,'DISCRIMINANT GENERALIZATION.');
( 27)   59  CHARACTERISTIC:
( 27)   60           WRITELN(OUTPUT,'CHARACTERISTIC GENERALIZATION, ', 'MINCOVER IS ',PRM.MINCOVER:2,'%');
( 27)   61           END;(* CASE PRM.DESCTYPE *)
( 27)   62        END;
( 28)    1  *DECK WTOLER
( 28)    2  PROCEDURE WTOLER
( 28)    3       (TOLER          : REAL);(* PURPOSE:
( 28)    4        TO CORRECTLY PRINT THE TOLERANCE.  IF TOLERANCE IS ABSOLUTE, PRINT
( 28)    5      AS A REAL NUMBER, IF RELATIVE, PRINT AS A PERCENT. *)
( 28)    6  BEGIN
( 28)    7        IF TOLER=TRUNC(TOLER) THEN WRITE(OUTPUT,TOLER:5:2)(* ABSOLUTE TOLERANCE *)
( 28)    8        ELSE WRITE(OUTPUT,ROUND(100*TOLER):4,'%');(* RELATIVE TOLERANCE *)
( 28)    9        END;(*WTOLER*)
( 29)    1  *DECK MAX
```

```
( 29)    2  FUNCTION MAX
( 29)    3       (I,J            : INTEGER): INTEGER;
( 29)    4  BEGIN
( 29)    5       IF I>J THEN MAX:=I ELSE MAX:=J;
( 29)    6       END;(*MAX*)
( 30)    1  *DECK GETNUM
( 30)    2  PROCEDURE GETNUM
( 30)    3       (VAR J          : INTEGER;        (* SCAN THE COMMAND FOR A NUMERIC FIELD *)
( 30)    4       VAR I           : INTEGER);       (* J: CURRENT BUFFER CURSOR POSITION *)
( 30)    5                                         (* I: RESULT OF NUMERIC SCAN *)
( 30)    6  LABEL
( 30)    7       1,
( 30)    8       2;
( 30)    9  VAR
( 30)   10       NEG             : BOOLEAN;
( 30)   11  BEGIN
( 30)   12       NEG:=FALSE;       I:=0;
( 30)   13       WHILE J<BLEN DO BEGIN
( 30)   14          J:=J+1;
( 30)   15          IF BUF[J]='-' THEN NEG:=TRUE;
( 30)   16          IF (BUF[J] IN ['0'..'9'])AND(NOT(BUF[J-1] IN ['0'..'9','X']))THEN GOTO 1;
( 30)   17          END;
( 30)   18  1:   WHILE (J<=BLEN)AND(BUF[J]IN['0'..'9'])DO BEGIN
( 30)   19          I:=I*10+ORD(BUF[J])-ORD('0');      J:=J+1;
( 30)   20          END;
( 30)   21       IF NEG THEN I:=-I;
( 30)   22       END;
( 31)    1  *DECK SOUT
( 31)    2  PROCEDURE SOUT;
( 31)    3  VAR
( 31)    4       I,J             : INTEGER;
( 31)    5  BEGIN
( 31)    6       STAB^:=S;                         (* PUT SYMTAB INTO BUFFER*)
( 31)    7       PUT(STAB);                             REWRITE(DFILE);
( 31)    8       DFILE^:=DST;                      (* PUT DOMAIN STRUCTURES INTO BUFFER *)
( 31)    9       PUT(DFILE);
( 31)   10       END; (* SOUT *)
( 32)    1  *DECK ADDCONS
( 32)    2  PROCEDURE ADDCONS
( 32)    3       (G1,G2          : GPTR);
( 32)    4  LABEL
( 32)    5       1,
( 32)    6       99;
( 32)    7  VAR
( 32)    8       I,J,K,L,M,N     : INTEGER;
( 32)    9  BEGIN  (* CONSEQUENCE IS IN GSIZE NODE *)
( 32)   10
( 32)   11       FOR J:=1 TO GSIZE DO              (* EXAMINE EACH NODE IN THE GRAPH *)
( 32)   12       IF (G1^.PNO[GSIZE]=G2^.PNO[J]) THEN  (* IF IT HAS SAME SYMBOL AS THE RESTRICTION NODE *)
( 32)   13          IF INSIDE(G1^.PNO[GSIZE],G2^.VAL[J],G1^.VAL[GSIZE],TRUE)
( 32)   14             THEN BEGIN(* AND ITS VALUES ARE A SUBSET OF THE RESTRICTIONS VALUES *)
( 32)   15             I:=1;                       (* CHECK ITS ISOMORPHISM POINTERS TO SEE IF IT POINTS *)
( 32)   16             WHILE G2^.LNK[J,I]<>0 DO    (* TO THE CONSEQUENCE OF RESTRICTION ALREADY *)
( 32)   17             BEGIN
( 32)   18                IF G2^.ASSGN[G2^.LNK[J,I]]<>G1^.LNK[GSIZE,I] THEN GOTO 1;(* IF SO, JUMP OUT *)
( 32)   19                I:=I+1;                  (* TRY NEXT LINK *)
( 32)   20                END;
( 32)   21             G2^.VAL[J]:=G1^.VAL[GSIZE];  (* DIDNT FIND IT, SO REPLACE CONSEQUENCE SET *)
```

```
( 32)   22                    (*CONSEQUENCE ALREADY IN G2, RETURN*)
( 32)   23
( 32)   24                GOTO 99;                    (* WE JUST REPLACE VALUE SET AND LEAVE *)
( 32)   25  1:            ;
( 32)   26                END;(*FOR J*)
( 32)   27
( 32)   28           (*CONSEQUENCE NOT IN G2, ADD TO G2*)
( 32)   29
( 32)   30           I:=1;                            (* FIND FIRST UNUSED NODE ENTRY *)
( 32)   31           WHILE G2^.LNK[I,1]<>0 DO BEGIN
( 32)   32              I:=I+1;
( 32)   33              IF I > GSIZE THEN WRITELN(OUTPUT,'ADDED RESTRICTIONS HAVE OVERFLOWED' ,' GRAPH SIZE--GSIZE');
( 32)   34              END;
( 32)   35           G2^.PNO[I]:=G1^.PNO[GSIZE];      (* COPY INFORMATION TO CREATE NEW SELECTOR *)
( 32)   36           G2^.VAL[I]:=G1^.VAL[GSIZE];      (* COPY SYMBOL NUMBER, VALUE SET *)
( 32)   37           G2^.VBL[I]:=G1^.VBL[GSIZE];      (* VBL FLAG *)
( 32)   38           G2^.ORDIRR[I]:=G1^.ORDIRR[GSIZE];(* ORDIRR FLAG *)
( 32)   39           J:=1;(*ADD SELECTOR TO G2*)
( 32)   40
( 32)   41           WHILE G1^.LNK[GSIZE,J]<>0 DO     (* ADD POINTERS TO GRAPH FOR THIS NODE *)
( 32)   42           BEGIN
( 32)   43              G2^.LNK[I,J]:=G1^.ASSGN[G1^.LNK[GSIZE,J]];      L:=1;(* FIND FIRST UNUSED LINK ENTRY *)
( 32)   44              WHILE G2^.LNK[G2^.LNK[I,J],L]<>0 DO BEGIN
( 32)   45                 L:=L+1;
( 32)   46                 IF L>MLNK THEN WRITELN(OUTPUT,'ADDED RESTRICTIONS HAVE OVERFLOWED' ,' GRAPH SIZE--MLNK');
( 32)   47                 END;(* WHILE *)
( 32)   48              G2^.LNK[G2^.LNK[I,J],L]:=I;      J:=J+1;
( 32)   49              END;
( 32)   50           IF CHRR='X' THEN BEGIN
( 32)   51              I:=1;(* EXCHANGE MODE, DELETE OLD PARTS OF GRAPH *)
( 32)   52              WHILE (I<GSIZE) DO BEGIN
( 32)   53                 IF (NOT G1^.VBL[I]) AND (G1^.ASSGN[I]<>0) THEN BEGIN
( 32)   54                    J:=G1^.ASSGN[I];(* FOR EACH SELECTOR (NOT VBL) *)
( 32)   55                    K:=1;(* INSPECT EACH OUTGOING LINK *)
( 32)   56                    WHILE (K<=MLNK) AND (G2^.LNK[J,K]<>0) DO BEGIN
( 32)   57                       L:=G2^.LNK[J,K];      M:=1;      N:=1;(* FIND CORRESPONDING INCOMING LINK *)
( 32)   58                       WHILE (M<=MLNK) AND (G2^.LNK[L,M]<>0) DO BEGIN
( 32)   59                          IF G2^.LNK[L,M]<>J THEN BEGIN(* COPY ONLY LINKS THAT DONT POINT TO SELECTOR *)
( 32)   60                             G2^.LNK[L,N]:=G2^.LNK[L,M];(* THUS DELETING INCOMING LINKS *)
( 32)   61                             N:=N+1;
( 32)   62                             END;
( 32)   63                          M:=M+1;
( 32)   64                          END;(* WHILE M *)
( 32)   65                       G2^.LNK[L,N]:=0;(* MAKE SURE LAST LINK IS 0 *)
( 32)   66                       G2^.LNK[J,K]:=0;(* DELETE THIS OUTGOING LINK TOO *)
( 32)   67                       K:=K+1;
( 32)   68                       END;(* WHILE K *)
( 32)   69                    G2^.COUNT[J]:=0;(* REMOVE NODE FROM GRAPH *)
( 32)   70                    END;(* IF NOT VBL *)
( 32)   71                 I:=I+1;
( 32)   72                 END;(* WHILE I, REMOVE THIS SELECTOR *)
( 32)   73              END;(* X MODE *)
( 32)   74  99:
( 32)   75           END;  (* ADDCONS *)
( 33)    1  *DECK ALLC
( 33)    2  PROCEDURE ALLC
( 33)    3        (VAR F1          : CPTR;
( 33)    4         GSUB,G          : GPTR);
```

```
( 33)    5  LABEL
( 33)    6        1,
( 33)    7        2;
( 33)    8  VAR
( 33)    9        I,J,K          : INTEGER;
( 33)   10        P              : CPTR;
( 33)   11  BEGIN
( 33)   12        NEWC(P);                              (* GET A NEW COMPLEX *)
( 33)   13        NEWCPXTAB(P^.TAB);                    (* AND GET AN ASSOCIATED TABLE *)
( 33)   14
( 33)   15        WITH P^.TAB^ DO BEGIN                 (* INIT TABLE *)
( 33)   16           COVCOUNT:=1;       COVLIST[1]:=G^.RNO;(* INDICATE THAT IT COVERS THIS RULE *)
( 33)   17           END;(* WITH *)
( 33)   18
( 33)   19        P^.NXTC:=F1;                          (* LINK NEW CPX TO F1 SET *)
( 33)   20        F1:=P;
( 33)   21        FOR J:=1 TO MST.NMST DO BEGIN         (* COPY THE METASELECTORS INTO THE COMPLEX *)
( 33)   22           F1^.CVAL[J]:=G^.MSEL^.CVAL[MST.PTR[J]];      AQP.SLOC[J]:=MST.SYMPTR[MST.PTR[J]];
( 33)   23           END;
( 33)   24        J:=MST.NMST;                          (* J IS # OF ENTRIES IN CPX USED *)
( 33)   25        FOR I:=1 TO GSIZE DO IF (GSUB^.COUNT[I]=1) THEN(* IF THIS NODE IS ACTIVE IN THIS ISOMORPHISM *)
( 33)   26        BEGIN
( 33)   27           J:=J+1;                            (* COPY NODE TO CPX *)
( 33)   28           AQP.SLOC[J]:=ABS(GSUB^.PNO[I]);    (* COPY PNO *)
( 33)   29           K:=GSUB^.ASSGN[I];                (* FIX COMPLEX EXPRESSION *)
( 33)   30           F1^.CVAL[J]:=G^.VAL[K];    (* AND ASSIGNED VALUES *)
( 33)   31           END;
( 33)   32        AQP.NVAR:=J;                          (* SET NUMBER OF VARS IN CPX *)
( 33)   33        P:=F1^.NXTC;                          (* EXAMINE THE REMAINDER OF F1 FOR DUPLICATES *)
( 33)   34        WHILE P<>NIL DO BEGIN
( 33)   35           FOR J:=1 TO AQP.NVAR DO IF P^.CVAL[J]<>F1^.CVAL[J] THEN(* CHECK, IS IT A DUPLICATE CPX ? *)
( 33)   36           GOTO 1;                            (* IF NOT, SKIP TO 1 *)
( 33)   37           WITH P^.TAB^ DO BEGIN              (* UPDATE THIS TABLE, ADD THIS RULE *)
( 33)   38              I:=1;                           (* CHECK TO SEE IF RNO IS ALREADY HERE *)
( 33)   39              WHILE (I<=COVCOUNT) AND (COVLIST[I]<>G^.RNO) DO I:=I+1;
( 33)   40              IF I>COVCOUNT THEN BEGIN
( 33)   41                 COVCOUNT:=COVCOUNT+1;       COVLIST[COVCOUNT]:=G^.RNO;(* ADD THIS RULE TO LIST *)
( 33)   42                 END;(* IF I>COVCOUNT *)
( 33)   43              END;(* WITH *)
( 33)   44           P:=F1;                             (* ELSE, REMOVE THE NEW CPX SINCE IT IS DUPLICATED *)
( 33)   45           F1:=F1^.NXTC;                      (* SKIP F1 DOWN ONE RULE *)
( 33)   46           P^.NXTC := NIL;                       FREECPX(P);(* FREE CPX AND ASSOCIATED TABLE *)
( 33)   47           GOTO 2;                            (* ALL DONE *)
( 33)   48  1:       P:=P^.NXTC;
( 33)   49           END;
( 33)   50  2:    ;
( 33)   51        END; (* ALLC *)
( 34)    1  *DECK CALCARITH
( 34)    2  PROCEDURE CALCARITH
( 34)    3       (G1,G2          : GPTR;
( 34)    4        A              : ARITHPTR);(* PURPOSE:  TO ADD AN ARITHMETIC DERIVED DESCRIPTOR (AS DEFINED IN A)
( 34)    5     TO THE RULE IN G2.  G1 CONTAINS GRAPH BUILT FROM A^ WHICH HAS BEEN
( 34)    6     MATCHES BY SUBG1.  SUBG1 CALLS THIS ROUTINE TO ADD THE ARITH GUY TO G2.
( 34)    7     G1 AND G2 MATCH THRU ASSGN.
( 34)    8      *)
( 34)    9  VAR
( 34)   10        XSTK           : ARRAY[1..MAXASTACK] OF INTEGER;(* EXECUTION STACK *)
( 34)   11        I,J,K,L,M,N    : INTEGER;            (* MISC INTEGERS, SEE MEANINGS BELOW *)
```

```
( 34)   12        DONE            : BOOLEAN;          (* USED TO INDICATE LOOP END *)
( 34)   13        XTOP            : INTEGER;          (* TOP OF XSTK *)
( 34)   14
( 34)   15  BEGIN
( 34)   16        XTOP:=0;                            (* INIT STACK TO 0 *)
( 34)   17        FOR I:=2 TO A^.SIZE DO BEGIN        (* STARTING WITH SECOND ENTRY (SKIP LEFT HAND FUNCTION) *)
( 34)   18           CASE A^.STACK[I].ACTION OF       (* PROCEED TO EXECUTE STACK CODE *)
( 34)   19
( 34)   20  NUMBER:       BEGIN
( 34)   21                   XTOP:=XTOP+1;       XSTK[XTOP]:=A^.STACK[I].ARGUMENT;(* PUSH NUMBER ONTO XSTK *)
( 34)   22                   END;(* NUMBER *)
( 34)   23
( 34)   24  ADD:          BEGIN
( 34)   25                   XSTK[XTOP-1]:=XSTK[XTOP]+XSTK[XTOP-1];(* ADD 2 TOP ELEMENTS, RESULT ON TOP *)
( 34)   26                   XTOP:=XTOP-1;
( 34)   27                   END;
( 34)   28
( 34)   29  SUBTRACT:     BEGIN
( 34)   30                   XSTK[XTOP-1]:=XSTK[XTOP-1]-XSTK[XTOP];       XTOP:=XTOP-1;
( 34)   31                   END;(* SUBTRACT *)
( 34)   32
( 34)   33  MULTIPLY:     BEGIN
( 34)   34                   XSTK[XTOP-1]:=XSTK[XTOP-1]*XSTK[XTOP];       XTOP:=XTOP-1;
( 34)   35                   END;(* MULTIPLY *)
( 34)   36
( 34)   37  DIVIDE:       BEGIN
( 34)   38                   XSTK[XTOP-1]:=XSTK[XTOP-1] DIV XSTK[XTOP];       XTOP:=XTOP-1;
( 34)   39                   END;(* DIVIDE *)
( 34)   40
( 34)   41  MODULO:       BEGIN
( 34)   42                   XSTK[XTOP-1]:=XSTK[XTOP-1] MOD XSTK[XTOP];       XTOP:=XTOP-1;
( 34)   43                   END;(* MODULO *)
( 34)   44
( 34)   45  MINUS:        XSTK[XTOP]:=-XSTK[XTOP];
( 34)   46
( 34)   47  FUNCT:        BEGIN
( 34)   48                   J:=0;                    (* LOCATE SMALLEST VALUE IN DOMAIN *)
( 34)   49                   WHILE NOT (J IN G2^.VAL[G1^.ASSGN[A^.STACK[I].ARGUMENT]]) AND (J <= MNVAL) DO J:=J+1;
( 34)   50                   IF J<=MNVAL THEN BEGIN
( 34)   51                      XTOP:=XTOP+1;       XSTK[XTOP]:=J;(* PUSH VALUE ONTO STACK *)
( 34)   52                      END;(* IF *)
( 34)   53                   END;(* FUNCT *)
( 34)   54                END;(*CASE ACTION *)
( 34)   55             END;(* FOR I *)
( 34)   56
( 34)   57
( 34)   58     (* ADD THE ENTRY IN A^.STACK[1] TO THE GRAPH G2 *)
( 34)   59
( 34)   60
( 34)   61     (* VARIABLE USAGE:
( 34)   62     (*
( 34)   63     (* I:  NEW ROW BEING ADDED TO G2 FOR THIS NEW DESCRIPTOR
( 34)   64     (* J:  INDEX TO THE DUMMY LIST FOR ROW I
( 34)   65     (* K:  INDEX INTO ARITHSTACK TO FIND CORRESPONDING DUMMIES TO DUMMY[J]
( 34)   66     (* L:  INDEX INTO DUMMY LIST FOR FUNCTION IN ROW K OF ARITHSTACK
( 34)   67     (* M:  EXISTING ROW IN G2 OF DUMMY VARIABLE CORRESPONDING TO DUMMY[J]
( 34)   68     (* N:  INDEX TO FIND EMPTY SLOT IN LNK OF ROW M FOR ADDING BACK POINTER TO ROW I.
( 34)   69     (*
```

```
( 34)   70     (* BASIC ASSUMPTION MADE DURING ARITH PROCESSING BOTH HERE AND IN VLINT:
( 34)   71     (*
( 34)   72     (* LEFT HAND AND RIGHT HAND SIDE OF ARITHDD SHARE AT LEAST ONE DUMMY VARIABLE
( 34)   73     (* FOR ALL DESCRIPTORS WITH >1 ARGUMENT, ORDER IS IMPORTANT (ORDIRR=FALSE);
( 34)   74     (* THUS THE ORDER OF ARGUMENTS IN A^.STACK[I].DUMMY[...] IS THE SAME
( 34)   75     (* AS THE ORDER OF THOSE ARGUMENTS IN G1 AND G2.
( 34)   76     (*)
( 34)   77
( 34)   78        IF (XSTK[1]>=0) AND (XSTK[1]<=MNVAL) THEN BEGIN(* SKIP IF BAD VALUE *)
( 34)   79           I:=1;                          (* FIND A NEW ROW IN G2 TO USE *)
( 34)   80           WHILE (I<=GSIZE) AND (NOT (G2^.PNO[I]=0)) DO I:=I+1;
( 34)   81           IF I>GSIZE THEN WRITELN(OUTPUT,'ADDED ARITHMETIC DERIVED DESCRIPTOR TOO BIG FOR GRAPH', ' (GSIZE=',GSIZE:3,')')
( 34)   82              ;
( 34)   83           G2^.PNO[I]:=A^.STACK[1].ARGUMENT;       (* SET UP ROW I OF G2 *)
( 34)   84           G2^.VBL[I]:=FALSE;             (* NOT A VARIABLE *)
( 34)   85           G2^.DPNO[I]:=0;                (* SO NO DPNO... *)
( 34)   86           G2^.VAL[I]:=[XSTK[1]];            (* PUT VALUE INTO GRAPH *)
( 34)   87
( 34)   88           J:=1;                          (* FOR EACH DUMMY VARIABLE OF THIS DESCRIPTOR, *)
( 34)   89                                          (* FIND A CORRESPONDING DUMMY VARIABLE ON THE RHS OF DESCRIPTOR EQUATION *)
( 34)   90                                          (* AND LOCATE IT IN G2 TO ALLOW SETTING UP LINKS *)
( 34)   91           WHILE (J<=MLNK) AND (NOT (A^.STACK[1].DUMMY[J]=0)) DO BEGIN
( 34)   92              DONE:=FALSE;                (* SEARCH UNTIL WE FIND ONE, THEN SET DONE TO TRUE *)
( 34)   93              K:=1;                       (* SKIP FIRST ROW OF A^.STACK SINCE THAT IS LHS OF = *)
( 34)   94              WHILE (K< A^.SIZE) AND (NOT DONE) DO BEGIN
( 34)   95                 K:=K+1;                  (* EXAMINE THIS ROW *)
( 34)   96                 IF A^.STACK[K].ACTION=FUNCT THEN BEGIN(* IF ITS A FUNCTION *)
( 34)   97                    L:=1;                 (* LOOK AT ITS DUMMIES *)
( 34)   98                    WHILE NOT ((L>MLNK) OR (A^.STACK[K].DUMMY[L]=A^.STACK[1].DUMMY[J]) OR (A^.STACK[K].DUMMY[L]=0)) DO
( 34)   99                       L:=L+1;
( 34)  100                    IF NOT ((L>MLNK) OR (A^.STACK[K].DUMMY[L]=0)) THEN BEGIN(* IF WE HAVE A LEGIT MATCH *)
( 34)  101                       M:=G2^.LNK[G1^.ASSGN[A^.STACK[K].ARGUMENT],L];(* THE ABOVE ASSIGNMENT REQUIRES SOME EXPLANATION: *)
( 34)  102                                          (* A^.STACK[K].ARGUMENT IS CHANGES (IN BUILDG) TO GIVE THE INDEX *)
( 34)  103                                          (* INTO G1 OF THE CORRESPONDING DESCRIPTOR NODE IN THE GRAPH *)
( 34)  104                                          (* WE FOLLOW ASSGN TO FIND THE CORRESPONDING NODE IN G2 AND EXAMINE*)
( 34)  105                                          (* ITS L'TH LINK TO FIND A POINTER TO OUR DESIRED DUMMY VARIABLE *)
( 34)  106
( 34)  107                       G2^.LNK[I,J]:=M;   (* POINT OUR J'TH DUMMY AT THIS GUY *)
( 34)  108                       N:=1;              (* NOW FIND AN EMPTY SLOT FOR A BACK POINTER *)
( 34)  109                       WHILE NOT ((N>MLNK) OR (G2^.LNK[M,N]=0)) DO N:=N+1;
( 34)  110                       IF N>MLNK THEN WRITELN(OUTPUT,'ADDED ARITHMETIC DERIVED DESCRIPTOR ' ,' OVERFLOWS GRAPH (MLNK=',
( 34)  111                          MLNK:3,')');
( 34)  112                       G2^.LNK[M,N]:=I;      DONE:=TRUE;(* CEASE THE K LOOP *)
( 34)  113                       END;(* IF WE'VE FOUND A DUMMY *)
( 34)  114                    END;(* IF STACK[K] WAS FUNCT *)
( 34)  115                 END;(* WHILE NOT DONE *)
( 34)  116              J:=J+1;
( 34)  117              END;(* WHILE STACK[1] STILL HAS NONZERO DUMMY *)
( 34)  118           END;(* IF XSTK[1] IS LEGAL *)
( 34)  119        END;(* CALCARITH *)
( 35)    1  *DECK SUBG1
( 35)    2  FUNCTION SUBG1
( 35)    3       (G1,G2          : GPTR;
( 35)    4        ALLSUBG        : INTEGER;
( 35)    5        VAR F          : CPTR;
( 35)    6        INSD           : BOOLEAN;
( 35)    7        A              : ARITHPTR): BOOLEAN;(* IF ALLSUBG=3, CALL CALCARITH TO PROCESS DERIVED DESCRIPTORS *)
( 35)    8                                    (* PASS ARITHPTR AS EXPRESSION TO PROCESS *)
```

```
LABEL
      99,
      1,
      2;
TYPE
      LAR              = ARRAY[1..600] OF INTEGER;
VAR
      L1,L2,PTR        : INTEGER;(*
                 I N D U C E - 1  --  SUBG1  --  SUBG
 *)
 (*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                                SUBG(G1,G2:GPTR;
          RECURSIVE ALGORITHM WHICH DETERMINES IF THE TREE WITH
     ROOT ND1 IS A SUBGRAPH OF THE GRAPH WITH ROOT ND2.
 @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*)
*CALL SUBG
    (*SUBG*)
 (*1
                I N D U C E - 1  --  SUBG1 (MAIN CODE)
 *)

    (* THIS PROCEDURE SEARCHES FOR STARTING NODES IN G2*)

BEGIN
      SUBG1:=FALSE;
      IF (G1^.MSEL<>NIL)AND(G2^.MSEL<>NIL) THEN(* IF THEY BOTH HAVE MS'S *)
      FOR L1:=1 TO MST.NMST DO
         IF NOT (G1^.MSEL^.CVAL[MST.PTR[L1]]>=G2^.MSEL^.CVAL[MST.PTR[L1]]) THEN GOTO 99;
                               (* AND G1'S MS VALUES ARE NOT A SUPERSET OF G2'S *)
     (* THEN QUIT BECAUSE THEY CAN'T BE SUBGRAPHS *)
      FOR L1:=1 TO GSIZE-1 DO BEGIN
         G1^.ASSGN[L1]:=0;            (* CLEAN G1'S ASSIGN VECTORS *)
         G2^.ASSGN[L1]:=0;
         END;(*PRESCAN TO FIND IF POSSIBLE CORRESPONDENCD*)

      FOR L1:=1 TO GSIZE-1 DO IF G1^.LNK[L1,1]<>0 THEN(* SEARCH ALL 1ST LINKS-MAKE SURE THEY MATCH SOMETHING *)
      BEGIN
         FOR L2:=1 TO GSIZE DO
            IF(G2^.LNK[L2,1]<>0)AND(G2^.ASSGN[L2]=0) THEN(* IF THIS IS A FUNCTION WITH UNASSIGNED DUMMY *)
            IF(G2^.PNO[L2]=G1^.PNO[L1]) THEN    (* AND IT POINTS TO THE SAME DESCRIPTOR *)
               IF INSIDE(G1^.PNO[L1],G2^.VAL[L2],G1^.VAL[L1],INSD)THEN   (* AND I COVER IT *)
               BEGIN
                  G2^.ASSGN[L2]:=1;    (* MARK IT AS A POTENTIAL MATCH *)
                  GOTO 2;
                  END;
         GOTO 99;                     (* IF NONE FOUND, IT IS IMPOSSIBLE *)
2:       ;
         END;
      FOR L2:=1 TO GSIZE DO G2^.ASSGN[L2]:=0;(* CLEAN G2'S ASSIGN TABLE AGAIN *)
      FOR L1:=1 TO GSIZE-1 DO
         IF (G1^.LNK[L1,1]<>0)AND(G1^.LNK[L1,2]<>0) AND(NOT G1^.ORDIRR[L1]) THEN GOTO 1;
                               (* TRY TO FIND A BINARY FUNC. WITH ORDER IMPT. *)
         FOR L1:=1 TO GSIZE-1 DO      (* IF THAT FAILS *)
         IF(G1^.LNK[L1,1]<>0)AND(NOT G1^.ORDIRR[L1])THEN GOTO 1;(* HOW ABOUT A UNARY FUNC. WITH ORDER IMPORTANT *)
      FOR L1:=1 TO GSIZE DO          (* AND IF THAT FAILS *)
      IF G1^.LNK[L1,1]<>0 THEN GOTO 1; (* HOW ABOUT ANY UNARY FUNC. *)
1:    FOR L2:=1 TO GSIZE DO          (* L2 IS FUNC. TO MATCH *)
      IF G2^.LNK[L2,1]<>0 THEN       (* IF IT HAS AN ARGUMENT *)
```

```
( 35)   67          IF G1^.PNO[L1]=G2^.PNO[L2] THEN (* AND IT IS THE SAME DESCRIPTOR *)
( 35)   68             IF INSIDE(G1^.PNO[L1],G2^.VAL[L2],G1^.VAL[L1],INSD) THEN(* AND OUR VALUES INTERSECT/COVER *)
( 35)   69                IF SUBG(G1,G2,L1,L2) THEN(* AND THERE IS A SUBGRAPH BETWEEN US *)
( 35)   70                BEGIN
( 35)   71                   SUBG1:=TRUE;          (* THEN I COVER IT *)
( 35)   72                   GOTO 99;
( 35)   73                   END
( 35)   74                ELSE BEGIN               (* IF IT ISNT A SUBGRAPH *)
( 35)   75                   FOR PTR:=1 TO GSIZE DO(* CLEAN THE ASSIGNMENTS *)
( 35)   76                   BEGIN
( 35)   77                      G1^.ASSGN[PTR]:=0;      G2^.ASSGN[PTR]:=0;
( 35)   78                      END;
( 35)   79                   PTR:=-2;
( 35)   80                   END;                  (* AND TRY AGAIN *)
( 35)   81  99:    ;
( 35)   82         END;(* SUBG1 *)
( 36)    1  *DECK SUBG
( 36)    2  FUNCTION SUBG
( 36)    3       (G1,G2          : GPTR;
( 36)    4        N1,N2          : INTEGER): BOOLEAN;
( 36)    5  LABEL
( 36)    6        1,
( 36)    7        2,
( 36)    8        3,
( 36)    9        4;
( 36)   10  VAR
( 36)   11        P,P1,I,J,LASTP : INTEGER;
( 36)   12        DONE           : BOOLEAN;
( 36)   13        FATHER,L1,L2,ND1,ND2: ARRAY[1..200] OF INTEGER;(*
( 36)   14                  I N D U C E - 1  --  SUBG1  --  SUBG  --  MATCH
( 36)   15    *)
( 36)   16  *CALL MATCH
( 36)   17       (*MATCH*)
( 36)   18    (*1
( 36)   19                  I N D U C E - 1  --  SUBG1  --  SUBG (MAIN CODE)
( 36)   20    *)
( 36)   21
( 36)   22  BEGIN
( 36)   23        G1^.ASSGN[N1]:=N2;               (* ASSIGN THESE NODES TO EACH OTHER *)
( 36)   24        G2^.ASSGN[N2]:=N1;                    SUBG:=FALSE;(* NOT A SUBGRAPH YET *)
( 36)   25        P:=1;                                 FATHER[1]:=0;
( 36)   26        L1[1]:=1;                        (* LINKS IN G1 *)
( 36)   27        L2[1]:=1;                        (* LINKS IN G2 *)
( 36)   28        ND1[1]:=N1;                      (* NODES IN G1 *)
( 36)   29        ND2[1]:=N2;                      (* NODES IN G2 *)
( 36)   30  2:    WHILE P<>0 DO                    (* WHILE THE BACKUP LIST IS NON-EMPTY *)
( 36)   31        BEGIN
( 36)   32           WHILE G1^.LNK[ND1[P],L1[P]]=0 DO (* IF WE'RE AT THE END OF LINKS OF ND1[P]...*)
( 36)   33           BEGIN
( 36)   34              P1:=FATHER[P];             (* GET THE FATHER *)
( 36)   35              IF P1=0 THEN BEGIN         (* IF FATHER IS 0, WE'RE DONE.  IT IS A SUBG *)
( 36)   36                 SUBG:=TRUE;
( 36)   37                 GOTO 1;
( 36)   38                 END
( 36)   39              ELSE                       (* PARENT IS NON-ZERO, *)
( 36)   40              BEGIN
( 36)   41                 L1[P]:=L1[P1]+1;        (* EXAMINE NEXT LINK OF PARENT *)
( 36)   42                 ND1[P]:=ND1[P1];        (* MOVE TO NODE OF PARENT *)
```

```
( 36)   43                IF G1^.ORDIRR[ND1[P]] THEN (* IF FATHER'S ORDIRR THEN *)
( 36)   44                L2[P]:=1                   (* SET NEXT LINK TO 1 *)
( 36)   45                ELSE L2[P]:=L1[P];         (* ELSE COPY L1'S LINK *)
( 36)   46                ND2[P]:=ND2[P1];           (* COPY THE ND2 FROM FATHER TO SON *)
( 36)   47                FATHER[P]:=FATHER[P1];     (* COPY THE FATHER FROM FATHER TO SON *)
( 36)   48                END;
( 36)   49             END;
( 36)   50          REPEAT
( 36)   51             DONE:=TRUE;
( 36)   52             IF P<>0 THEN
( 36)   53                IF G2^.LNK[ND2[P],L2[P]]=0 THEN(* IF WE ARE AT THE END OF G2'S LINKS *)
( 36)   54                BEGIN
( 36)   55                   DONE:=FALSE;            (* SET NOT DONE *)
( 36)   56                   IF L1[P]=1 THEN         (* IF L1'S FIRST LINK  *)
( 36)   57                   BEGIN
( 36)   58                      G1^.ASSGN[ND1[P]]:=0; (* DEASSIGN THE NODE *)
( 36)   59                      G2^.ASSGN[ND2[P]]:=0;
( 36)   60                      END;
( 36)   61                   P:=P-1;                 (* BACKUP P *)
( 36)   62                   IF P<>0 THEN
( 36)   63                      IF NOT G1^.ORDIRR[ND1[P]] THEN L2[P]:=MLNK(* START WITH THE HIGHEST LINK *)
( 36)   64                      ELSE L2[P]:=L2[P]+1;(* UNLESS ORDIRR, THEN USE NEXT LNK *)
( 36)   65                   END;
( 36)   66             UNTIL DONE;
( 36)   67          IF P<>0 THEN                     (* NOW CHECK OUT THE NODES FOR MATCH *)
( 36)   68          CASE MATCH(P) OF
( 36)   69  0:             IF G1^.ORDIRR[ND1[P]] THEN  (* THEY DONT MATCH *)
( 36)   70                 L2[P]:=L2[P]+1             (* TRY NEXT L2 LNK *)
( 36)   71                 ELSE L2[P]:=MLNK;          (* UNLESS ORDER IS IMPT, TRY MLNK *)
( 36)   72  1:             BEGIN                      (* THEY ALREADY MATCH *)
( 36)   73                    LASTP:=P;               (* TRY NEXT LINK FROM THIS NODE *)
( 36)   74                    P:=P+1;                 (* REMEMBER LASTP AND ADD NEW LINE TO LIST *)
( 36)   75                    FATHER[P]:=FATHER[P-1]; (* THIS NODE HAS SAME FATHER *)
( 36)   76                    L1[P]:=L1[P-1]+1;       (* IT IS NEXT LINK OVER *)
( 36)   77                    ND1[P]:=ND1[P-1];       (* SAME NODE1 *)
( 36)   78                    IF G1^.ORDIRR[ND1[P]] THEN L2[P]:=1(* IF NO ORDER, START WITH FIRST *)
( 36)   79                    ELSE L2[P]:=L1[P];      (* ELSE KEEP IN STEP W/ L2 *)
( 36)   80                    ND2[P]:=ND2[P-1];
( 36)   81                    END;
( 36)   82  2:             BEGIN                      (* WE GOT A NEW MATCH *)
( 36)   83                    LASTP:=P;               (* REMEMBER LASTP *)
( 36)   84                    P:=P+1;                 (* ADD ANOTHER LINE TO BACKUP LIST *)
( 36)   85                    FATHER[P]:=P-1;         (* NEW FATHER *)
( 36)   86                    ND1[P]:=G1^.LNK[ND1[P-1],L1[P-1]]; (* FIGURE OUT WHICH NODES MATCHED *)
( 36)   87                    ND2[P]:=G2^.LNK[ND2[P-1],L2[P-1]];
( 36)   88                    L1[P]:=1;               (* AND LOOK AT THEIR FIRST LINKS *)
( 36)   89                    L2[P]:=1;
( 36)   90                    END
( 36)   91                 END;(*CASE STMT*)
( 36)   92
( 36)   93             END;(*WHILE*)
( 36)   94
( 36)   95  1:      IF (ALLSUBG<>0) AND (P<>0) THEN  (* SPECIAL CASE LOGIC-- *)
( 36)   96          BEGIN                           (* IF ALLSUBG<>0 WE ARE SUPPOSED TO FIND ALL SUBGRAPHS *)
( 36)   97             P:=LASTP;                    (* SO WE HAVE TO PICKUP AT LASTP AND KEEP ON*)
( 36)   98             IF G1^.ORDIRR[ND1[P]] THEN   (* BASICALLY DO A "CASE 0" PRETEND LAST NODE MISMATCHED *)
( 36)   99             L2[P]:=L2[P]+1
( 36)  100             ELSE L2[P]:=MLNK;
```

```
( 36)  101          SUBG:=FALSE;
( 36)  102          CASE ALLSUBG OF                 (* CALL APPROPRIATE ROUTINE TO PROCESS THESE ISOMORPHISMS *)
( 36)  103  1:         ADDCONS(G1,G2);              (* ADD CONSEQUENCES OF RESTRICTIONS *)
( 36)  104  2:         ALLC(F, G1, G2);             (* BUILD COMPLEX FOR AQ ROUTINE *)
( 36)  105  3:         CALCARITH(G1,G2,A);          (* COMPUTE ARITHMETIC DERIVED DESCRIPTORS *)
( 36)  106             END;(*CASE ALLSUBG *)
( 36)  107          GOTO 2;
( 36)  108          END;
( 36)  109       END;
( 37)    1  *DECK MATCH
( 37)    2  FUNCTION MATCH
( 37)    3       (P              : INTEGER): INTEGER;(* SEE IF NODES[P] MATCH *)
( 37)    4  VAR
( 37)    5       N1,N2,TMATCH,I,J: INTEGER;
( 37)    6  BEGIN
( 37)    7       N1:=G1^.LNK[ND1[P],L1[P]];      (* N1 IS NODE IN E1 *)
( 37)    8       N2:=G2^.LNK[ND2[P],L2[P]];      (* N2 IS NODE IN E2 TO MATCH WITH *)
( 37)    9       TMATCH:=0;                      (* SET NO MATCH INITIALLY *)
( 37)   10       IF (G1^.ASSGN[N1]=N2)AND(G2^.ASSGN[N2]=N1)THEN(* IF THEY ALREADY POINT TO EACH OTHER *)
( 37)   11       TMATCH:=1                       (* THEY MUST MATCH--WEAKLY *)
( 37)   12       ELSE IF(G1^.ASSGN[N1]=0)AND(G2^.ASSGN[N2]=0)THEN(* IF THEY ARE BOTH UNASSIGNED *)
( 37)   13          IF G1^.PNO[N1]=G2^.PNO[N2] THEN (* AND THEY POINT TO THE SAME DESCRIPTOR *)
( 37)   14             IF INSIDE(G1^.PNO[N1],G2^.VAL[N2],G1^.VAL[N1],INSD) THEN(* AND N1 HAS VALUES INSIDE N2 *)
( 37)   15             TMATCH:=2;                (* THEN THEY MATCH--STRONGLY *)
( 37)   16       IF TMATCH>0 THEN                (* BUT... LETS CHECK ABOUT THE ORDER OF THE LINKS *)
( 37)   17          IF NOT G1^.ORDIRR[N1] THEN   (* IF ORDER IS IMPORTANT *)
( 37)   18          BEGIN
( 37)   19             I:=1;
( 37)   20             WHILE G1^.LNK[N1,I]<>ND1[P] DO(* FIND THE LINK IN G1 TO ND1[P] *)
( 37)   21             I:=I+1;      J:=1;
( 37)   22             WHILE G2^.LNK[N2,J]<>ND2[P] DO(* FIND THE LINK IN G2 TO ND2[P] *)
( 37)   23             J:=J+1;
( 37)   24             IF I<>J THEN             (* THEY BETTER MATCH *)
( 37)   25             TMATCH:=0;               (* NO LUCK--THEY DON'T MATCH DUE TO ORDER *)
( 37)   26             END;
( 37)   27       IF TMATCH=2 THEN                (* IF THIS IS A NEW MATCH *)
( 37)   28       BEGIN
( 37)   29          G1^.ASSGN[N1]:=N2;           (* ASSIGN THE NODES TO EACH OTHER *)
( 37)   30          G2^.ASSGN[N2]:=N1;
( 37)   31          END;
( 37)   32       MATCH:=TMATCH;                  (* AND RETURN THE VALUE OF TMATCH: *)
( 37)   33                                       (* 1=ALREADY MATCHED BEFORE 2=NEW MATCH 0=NO MATCH *)
( 37)   34       END;
( 38)    1  *DECK PCPX
( 38)    2  PROCEDURE PCPX
( 38)    3       (F              : CPTR);        (* PRINT A VL1 COMPLEX AS A VL1 EXPRESSION *)
( 38)    4  VAR
( 38)    5       I,J,NSEL        : INTEGER;
( 38)    6  BEGIN
( 38)    7       NSEL:=1;
( 38)    8       FOR I:=1 TO AQP.NVAR DO IF F^.CVAL[I]<>[0..MNVAL] THEN(* IF NOT A =* REFERENCE *)
( 38)    9       BEGIN
( 38)   10          IF I>9 THEN                  (* PRINT OUT THE SUBSCRIPT *)
( 38)   11          WRITE(OUTPUT,'[X',I:2,'=')
( 38)   12          ELSE                         (* IN THE FORM [XI= *)
( 38)   13          WRITE(OUTPUT,'[X',I:1,'=');
( 38)   14          IF F^.CVAL[I]=[0..MNVAL] THEN   (* IF REFERENCE IS *, PRINT * *)
( 38)   15          WRITE(OUTPUT,'*')
```

```
( 38)          ELSE FOR J:=S.MVAL[ABS(AQP.SLOC[I])] TO S.EVAL[ABS( AQP.SLOC[I])] DO
( 38)             IF J IN F^.CVAL[I] THEN(* GO THRU DOMAIN, PRINT ALL VALUES IN REFERENCE *)
( 38)             WRITE(OUTPUT,J:3);      WRITE(OUTPUT,']');(* CLOSE THE SELECTOR *)
( 38)          NSEL:=NSEL+1;
( 38)          IF NSEL>8 THEN                  (* PRINT ONLY 8 SELECTORS TO A LINE *)
( 38)          BEGIN
( 38)             WRITELN(OUTPUT);      NSEL:=1;
( 38)             FOR J:=1 TO 5 DO             (* INDENT 5 BLANKS *)
( 38)             WRITE(OUTPUT,' ');
( 38)             END;
( 38)          END;(*FOR I:=*)
( 38)
( 38)       WRITELN(OUTPUT);
( 38)       IF PRM.DESCTYPE=CHARACTERISTIC THEN BEGIN
( 38)          IF F^.TAB<>NIL THEN BEGIN
( 38)             WRITE(OUTPUT,'COVERS RULES ');
( 38)             FOR I:=1 TO F^.TAB^.COVCOUNT DO WRITE(OUTPUT,F^.TAB^.COVLIST[I]:3);
( 38)             WRITELN(OUTPUT);
( 38)             END;(*IF F^.TAB<>NIL*)
( 38)          END;(* IF CHARACTERISTIC *)
( 38)       END;  (* PCPX *)
( 39) *DECK AQ
( 39) FUNCTION AQ
( 39)       (GSUB            : GPTR;            (* GSUB : CURRENT GRAPH BEGIN PROCESSED *)
( 39)       VL1M             : BOOLEAN;         (* VL1M : VL1 MODE FLAG, TRUE IF IN VL1 MODE *)
( 39)       F1,F2            : CPTR): CPTR;     (* F1 : SET OF F1 COMPLEXES TO COVER, F2 : SET OF FO COMPLEXES TO AVOID *)
( 39) LABEL
( 39)       1,
( 39)       2,
( 39)       3,
( 39)       4,
( 39)       10,
( 39)       12,
( 39)       13,
( 39)       7,
( 39)       8,
( 39)       21,
( 39)       22,
( 39)       23,
( 39)       99;                                       (* UGGH *)
( 39) VAR
( 39)       DELTA,I,J,K,L   : INTEGER;
( 39)       TEMPSET         : VALTP;(*  TEMP TO GET AROUND COMPILER BUG *)
( 39)       NSTAR,E1,E2,AQT,OSTAR,P,Q,R: CPTR;(*1
( 39)
( 39)                I N D U C E - 1  --  AQ  --  TRIM
( 39)
( 39)  *)
( 39)  (*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
( 39)                                   TRIM(VAR NSTAR:CPTR;
( 39)     TRIM NSTAR TO MAXS ELEMENTS
( 39)  @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*)
( 39) *CALL TRIM
( 39)    (*TRIM*)
( 39)  (*1
( 39)                I N D U C E - 1  --  AQ(MAIN CODE)
( 39)  *)
( 39)
```

```
39)   38  BEGIN  (* PLACE ALL EVENTS INTO FQ AND FP SETS *)
39)   39
39)   40        AQ:=NIL;                          (* NO VALUE YET *)
39)   41        IF (F1=NIL) THEN                  (* IF NOTHING TO DO, QUIT *)
39)   42        GOTO 99;
39)   43        WITH AQP DO                       (* REFER TO AQP PARAMETERS FOR REMAINDER OF PROCEDURE *)
39)   44        BEGIN
39)   45           AQT:=NIL;                      (* NO RESULT SET YET *)
39)   46           P:=F1;                         (* SET THE FP AND FQ FLAGS FOR ALL F1 *)
39)   47           WHILE P<>NIL DO BEGIN
39)   48              P^.FP:=TRUE;       P^.FQ:=TRUE;       P:=P^.NXTC;
39)   49              END;(* ALLOCATE START OF OSTAR *)
39)   50
39)   51           DELTA:=1;                      (* FIRST PASS, DELTA=1 *)
39)   52  1:       NSTAR:=NIL;
39)   53           NEWC(OSTAR);                   (* GET A COMPLEX, PUT IT ON OSTAR *)
39)   54           OSTAR^.NXTC:=NIL;     OSTAR^.FP:=TRUE;(* PUT IT INTO FP SET *)
39)   55           FOR I:=1 TO AQP.NVAR DO        (* SET ALL VARIABLES TO *--THIS COMPLEX COVERS ENTIRE EVENT SET *)
39)   56           OSTAR^.CVAL[I]:=[0..MNVAL];(* FIND UNCOVERED EVENT *)
39)   57
39)   58           E1:=F1;
39)   59  13:       IF NOT (((DELTA=1) AND (E1^.FP))OR((DELTA=2) AND (E1^.FQ)))
39)   60     THEN BEGIN(* IF DELTA= 1, FIND FIRST E1 RULE WITH FP SET *)
39)   61              E1:=E1^.NXTC;               (* IF DELTA= 2, FIND FIRST E1 RULE WITH FQ SET *)
39)   62              IF NOT VL1M THEN            (* IN VL1 MODE, WE GO UNTIL E1 IS EXHAUSTED *)
39)   63              GOTO 12;                    (* OTHERWISE, QUIT *)
39)   64              IF E1=NIL THEN GOTO 12 ELSE GOTO 13;
39)   65              END;
39)   66           E2:=F2;                        (* EXAMINE EACH RULE IN THE OPPONENT SET (F2) *)
39)   67           WHILE E2<>NIL DO BEGIN(* SEE IF E2 IS IN OSTAR *)
39)   68
39)   69              P:=OSTAR;                   (* FOR EACH ELEMENT IN OSTAR *)
39)   70              WHILE P<>NIL DO BEGIN
39)   71                 FOR I:=1 TO AQP.NVAR DO   (* EXAMINE THEIR VALUE SETS, IF THEY INTERSECT *)
39)   72                 IF (E2^.CVAL[I]*P^.CVAL[I])=[] THEN GOTO 2;(* JUMP OUT--IF THEY DON'T INTERSECT *)
39)   73                 GOTO 3;
39)   74  2:              P:=P^.NXTC;             (* IF THEY DON'T INTERSECT, GREAT, TRY ANOTHER *)
39)   75                 END;(* WHILE P<>NIL*)
39)   76
39)   77              GOTO 10;                    (* NOTHING IN OSTAR INTERSECTS THIS RULE, THEYRE CONSISTENT *)
39)   78                    (* E2 IS IN OSTAR, FINE ELEMENTARY STAR OF E1 AGAI
39)   79                     NST E2 *)
39)   80
39)   81                    (* MULTIPLY BY OSTAR *)
39)   82
39)   83  3:           FOR I:=1 TO AQP.NVAR DO     (* WE HAVE AN INCONSISTENCY BETWEEN E2 AND OSTAR *)
39)   84              IF E1^.CVAL[I]<=([0..MNVAL]- E2^.CVAL[I]) THEN    (* FIND A VARIABLE OF E1 *)
39)   85              BEGIN                       (* WHICH IS CONSISTENT WITH E2 VARIABLE *)
39)   86                 P:=OSTAR;     (* PUT CPX FROM OSTAR INTO NSTAR, MPY BE E2 COMPL
39)   87                               *)
39)   88
39)   89                 WHILE P<>NIL DO          (* FOR EACH CPX ON OSTAR, MAKE A COPY ON NSTAR *)
39)   90                 BEGIN
39)   91                    NEWC(R);              (* GET A NEW CPX TO MAKE COPY *)
39)   92                    R^.NXTC:=NSTAR;                NSTAR:=R;
39)   93                    FOR J:=1 TO AQP.NVAR DO      (* COPY THE VALUES *)
39)   94                    R^.CVAL[J]:=P^.CVAL[J];      FIXIT:=R^.CVAL[I];      EXTND (AQP.SLOC[I],E1^.CVAL[I], E2^.CVAL[I]);
39)   95                    R^.CVAL[I]:=FIXIT;      (* EXTENT THE REFERENCES OF THIS E1 VARIABLE AGAINST E2 AND THEN *)
```

```
( 39)   96                  P:=P^.NXTC;          (* SHRINK THE VALUE SET ON THE NSTAR--SEE EXTND CODE *)
( 39)   97                  END;      (* WHILE P<>NIL *)
( 39)   98
( 39)   99               END;(* FOR I *)
( 39)  100
( 39)  101                   (* NOW APPLY ABSOURPTION LAWS TO NSTAR *)
( 39)  102
( 39)  103            P:=NSTAR;
( 39)  104            WHILE P<>NIL DO            (* SET ALL NSTAR FP'S TO TRUE *)
( 39)  105            BEGIN
( 39)  106               P^.FP:=TRUE;       P:=P^.NXTC;
( 39)  107               END;
( 39)  108            P:=NSTAR;
( 39)  109            WHILE P<> NIL DO           (* GO THRU NSTAR BUBBLE STYLE *)
( 39)  110            BEGIN
( 39)  111               IF P^.FP THEN           (* IF THIS RULE IS STILL "IN" *)
( 39)  112               BEGIN
( 39)  113                  Q:=NSTAR;
( 39)  114                  WHILE Q<>NIL DO BEGIN
( 39)  115                     IF Q^.FP AND (Q<>P) THEN     (* AND THIS RULE IS STILL "IN" AND THEYRE DIFFERENT *)
( 39)  116                     BEGIN             (* AND THERE IS AT LEAST ONE VARIABLE WHERE P DOESNT*)
( 39)  117                        FOR I:=1 TO AQP.NVAR DO      (* COVER Q *)
( 39)  118                        IF NOT(Q^.CVAL[I]<=R^.CVAL[I]) THEN        (* IF THIS AN ERROR????*)
( 39)  119                        GOTO 4;           (* THEN JUMP OUT *)
( 39)  120                        Q^.FP:=FALSE;       (* OTHERWISE, IF P COVERS Q, REMOVE FROM FP *)
( 39)  121                        END;          (*IF Q^.FP*)
( 39)  122
( 39)  123  4:                    Q:=Q^.NXTC;       (* MOVE ON TO NEXT Q RULE *)
( 39)  124                        END;      (*WHILE Q<>NIL*)
( 39)  125
( 39)  126                     END;   (* IF P^.FP *)
( 39)  127
( 39)  128               P:=P^.NXTC;
( 39)  129               END;(*WHILE P*)
( 39)  130
( 39)  131                   (* ABSOURPTION COMPLETE *)
( 39)  132
( 39)  133                   (* TRIM NUMBER OF COMPLEXES *)
( 39)  134
( 39)  135            TRIM(NSTAR,AQP.MAXSTARAQ);  (* EVALUATE COSTS, TRIM TO MAXSTAR BEST ELEMENTS *)
( 39)  136                   (*RETURN OLIST TO AQP.FREEC *)
( 39)  137
( 39)  138            IF NSTAR=NIL THEN          (* IF NOTHING LEFT, QUIT *)
( 39)  139            GOTO 10;
( 39)  140            FREECPX(OSTAR);            (* RELEASE ALL RULES IN OSTAR *)
( 39)  141            OSTAR:=NSTAR;              (* MAKE THE OSTAR THE NSTAR *)
( 39)  142            NSTAR:=NIL;                (* AND RESET NSTAR *)
( 39)  143  10:       E2:=E2^.NXTC;              (* BUILD ANOTHER NSTAR FROM NEXT E2 RULE *)
( 39)  144            END;(* WHILE E2<>NIL *)
( 39)  145
( 39)  146             (* UPDATE FP AND FQ SETS *)
( 39)  147
( 39)  148         P:=OSTAR;
( 39)  149         WHILE P<>NIL DO               (* REMOVE ALL RULES WHICH ARE NOW COVERED BY OSTAR *)
( 39)  150         BEGIN
( 39)  151            Q:=F1;
( 39)  152            WHILE Q<>NIL DO            (* FROM F1 SET *)
( 39)  153            BEGIN
```

```
 39)  154                IF Q^.FP THEN FOR I:=1 TO AQP.NVAR DO
 39)  155                   IF NOT (Q^.CVAL[I]<=P^.CVAL[I]) THEN GOTO 7;
 39)  156                Q^.FP:=FALSE;
 39)  157  7:             Q:=Q^.NXTC;
 39)  158                END;(* WHILE Q<>NIL*)
 39)  159
 39)  160             P:=P^.NXTC;
 39)  161             END;(* WHILE P<>NIL *)
 39)  162
 39)  163              (* FINE NEXT F1 TO COVER *)
 39)  164
 39)  165          IF OSTAR=NIL THEN              (* IF NO OSTAR AFTER ALL OF THIS, REMOVE E1 FROM F1 *)
 39)  166          BEGIN
 39)  167             E1^.FP:=FALSE;       E1^.FQ:=FALSE;
 39)  168             GOTO 1;                    (* AND GO FIND ANOTHER E1 *)
 39)  169             END;
 39)  170          TRIM(OSTAR,1);                (* GET BEST ELEMENT OF OSTAR *)
 39)  171          P:=OSTAR;                     (* AND RETURN REST OF OSTAR TO FREEC *)
 39)  172              (*LQST*)
 39)  173
 39)  174          IF AQP.FREEC=NIL THEN         (* GET A NEW COMPLEX AT TOP OF FREEC QUEUE *)
 39)  175          BEGIN
 39)  176             NEW(AQP.FREEC);            (* WE USE THIS COMPLEX AS A TEMPORARY VARIABLE--CALL IT AQP *)
 39)  177             AQP.FREEC^.TAB:=NIL;       (* NO ASSOCIATED TABLE *)
 39)  178             AQP.FREEC^.NXTC:=NIL;
 39)  179             END;
 39)  180          FOR I:=1 TO AQP.NVAR DO
 39)  181             IF (F2=NIL)OR(P^.CVAL[I]<>[0..MNVAL]) THEN(* INITIALIZE AQP TO A SUBSET OF EVENT SPACE *)
 39)  182             AQP.FREEC^.CVAL[I]:=[]     (* FOR EVERY NON-* VARIABLE OF OSTAR, SET AQP TO NULL SET *)
 39)  183          ELSE AQP.FREEC^.CVAL[I]:=[0..MNVAL];(* BUT IF OSTAR IS *, THEN AQP IS ALSO * *)
 39)  184          Q:=F1;                        (* GO THRU F1 AND MARK FQ FOR EACH RULE COVERED BY OSTAR *)
 39)  185          WHILE Q<>NIL DO BEGIN
 39)  186             FOR I:=1 TO AQP.NVAR DO IF NOT (Q^.CVAL[I]<=P^.CVAL[I]) THEN(* DOES OSTAR COVER IT? *)
 39)  187             GOTO 8;                    (* NO, JUMP OUT *)
 39)  188             Q^.FQ:=FALSE;              (* YES, SET FQ TO FALSE*)
 39)  189             FOR I:= 1 TO AQP.NVAR DO BEGIN (* "OR" IN ALL OF THESE "COVERED" VALUES INTO AQP *)
 39)  190                TEMPSET := AQP.FREEC^.CVAL[I];      TEMPSET := TEMPSET + Q^.CVAL[I];
 39)  191                AQP.FREEC^.CVAL[I]:= TEMPSET;  (* GET AROUND COMPILER BUG? *)
 39)  192                END;
 39)  193  8:          Q:=Q^.NXTC;
 39)  194             END;(* WHILE Q<>NIL*)
 39)  195
 39)  196          OSTAR^.NXTC:=AQT;             (* PUT THIS OSTAR CPX ON AQT LIST OF GOOD CPX'S TO RETURN *)
 39)  197          AQT:=OSTAR;
 39)  198          IF AQP.LQST THEN              (* IF LQST IS SET, PROCESS THIS AQP CPX TO TRIMM IT *)
 39)  199          BEGIN
 39)  200             FOR I:=1 TO AQP.NVAR DO    (* FOR EACH VARIABLE *)
 39)  201             CASE S.VTYPE[AQP.SLOC[I]] OF
 39)  202  1:                ;                   (* IF NOMINAL, NO PROBLEM *)
 39)  203  2:                BEGIN               (* IF INTERVAL, *)
 39)  204                      FOR J:=0 TO MNVAL DO  (* FIND LOWEST ELEMENT IN VALUE SET *)
 39)  205                      IF J IN AQP.FREEC^.CVAL[I] THEN GOTO 21;
 39)  206  21:                FOR K:=MNVAL DOWNTO 0 DO  (* FIND HIGHEST ELEMENT IN VALUE SET *)
 39)  207                      IF K IN AQP.FREEC^.CVAL[I] THEN GOTO 22;
 39)  208  22:                FOR L:=J TO K DO      (* ADD ALL VALUES INBETWEEN LOWEST ANDHIGHEST *)
 39)  209                      AQP.FREEC^.CVAL[I]:=AQP.FREEC^.CVAL[I] + [L];
 39)  210                      END;
 39)  211  3:                BEGIN               (* IF STRUCTURED DOMAIN *)
```

```
( 39)              IF F2<>NIL THEN           (* AND THERE IS SOMETHING TO COVER AGAINST *)
( 39)              AQP.FREEC^.CVAL[I]:=OSTAR^.CVAL[I]    (* SIMPLY COPY THE GENERALIZATION IN OSTAR *)
( 39)              ELSE                      (* BUT IF THERE IS NO OPPONENT SET *)
( 39)              FOR J:=1 TO DST.NELE DO     (* ONLY GENERALIZE IF ALL ELEMENTS OF VALUE SET *)
( 39)              IF DST.PNO[J]=AQP.SLOC[I] THEN      (* FIT A SINGLE PREMISE OF GSTRUC *)
( 39)                IF AQP.FREEC^.CVAL[I]<=DST.PREM[J] THEN BEGIN
( 39)                  AQP.FREEC^.CVAL[I]:=AQP.FREEC^.CVAL[I]+DST.CONS[J];
( 39)                  GOTO 23;          (* WE ONLY GENERALIZE 1 LEVEL UNDER THIS SCHEME *)
( 39)                  END;
( 39)              END
( 39)            END;(*CASE STMT*)
( 39)
( 39) 23:      FOR I:=1 TO AQP.NVAR DO     (* NEW COPY THE CVALS BACK TO OSTAR CPX *)
( 39)          OSTAR^.CVAL[I]:=AQP.FREEC^.CVAL[I];
( 39)          END;(*LQST*)
( 39)
( 39)        GOTO 1;                     (* GO DO IT AGAIN *)
( 39)           (* PASS 2 *)
( 39)
( 39) 12:    IF DELTA = 1 THEN             (* SET DELTA=2 AFTER FIRST PASS THRU F1 *)
( 39)        BEGIN
( 39)          DELTA:=2;
( 39)          GOTO 1;                   (* AND GO CHECK FQ FLAGS *)
( 39)          END;(* FIND BEST COMPLEX IN COVER *)
( 39)
( 39)        P:=AQT;                     (* SET ALL FP'S IN AQT SO TRIM WILL EXAMINE THEM *)
( 39)        WHILE P<> NIL DO BEGIN
( 39)          P^.FP:=TRUE;      P:=P^.NXTC;
( 39)          END;
( 39)        IF NOT VL1M THEN            (* IF NOT IN VL1 MODE, TRIM THE LIST TO 1 COMPLEX *)
( 39)        TRIM(AQT,1);      AQ:=AQT;    (* AND RETURN IT *)
( 39)        END;(*WITH AQP*)
( 39)
( 39) 99:
( 39)      END; (* AQ PROCEDURE *)
( 40) *DECK TRIM
( 40) PROCEDURE TRIM
( 40)    (VAR NSTAR      : CPTR;
( 40)      MAXS          : INTEGER);
( 40) LABEL
( 40)      1,
( 40)      2,
( 40)      99;
( 40) TYPE
( 40)      ATYPE         = ARRAY[0..300] OF CPTR;
( 40) VAR
( 40)      CA            : ATYPE;
( 40)      X             : REAL;
( 40)      NC,I,J,IB,IC  : INTEGER;(*
( 40)           I N D U C E - 1  --  AQ  --  TRIM  --  COSTF
( 40)
( 40)  *)
( 40)  (*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
( 40)                     COSTF(P:CPTR;
( 40)        DETERMINE COST OF THIS COMPLEX
( 40)  @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*)
( 40) *CALL COSTF
( 40)    (*COSTF*)
```

```
( 40)   24   (*1
( 40)   25                  I N D U C E - 1  --  AQ  --  TRIM (MAIN CODE)
( 40)   26   *)
( 40)   27
( 40)   28  BEGIN  (*TRIM*)
( 40)   29
( 40)   30        IC:=1;       IB:=1;       P:=NSTAR;       NC:=0;
( 40)   31        WHILE P<>NIL DO BEGIN
( 40)   32           Q:=P;
( 40)   33           IF P^.FP THEN BEGIN            (* IF UNDER CONSIDERATION, PUT ITS POINTER IN "CA" ARRAY *)
( 40)   34              NC:=NC+1;       CA[NC]:=P;       P:=P^.NXTC;
( 40)   35              END(*IF P^.FP*)
( 40)   36
( 40)   37           ELSE BEGIN                     (* ELSE, RETURN THIS CPX TO FREE LIST *)
( 40)   38              P:=P^.NXTC;       Q^.NXTC:=AQP.FREEC;       FREETAB(Q^.TAB);(* FREE ASSOCIATED TABLE *)
( 40)   39              AQP.FREEC:=Q;
( 40)   40              END;
( 40)   41           END;(*WHILE P<>NIL *)
( 40)   42
( 40)   43        CA[NC+1]:=CA[NC];              (* EXTENT THE ENDS OF THE ARRAY FOR THIS ALGORITHM *)
( 40)   44        CA[0]:=CA[1];
( 40)   45  1:    IF NC<=MAXS THEN               (* IF WE DONT HAVE TO TRIM, QUIT *)
( 40)   46        GOTO 99;
( 40)   47        I:=1;
( 40)   48        IF MAXS=0 THEN GOTO 2;         (*??*)
( 40)   49        FOR J:=1 TO NC DO              (* EVALUATE COST FUNCTIONS *)
( 40)   50        CA[J]^.COST:=COSTF(CA[J],AQP.CSTF[IC]);(*SORT ARRAY CA *)
( 40)   51
( 40)   52        FOR I:=IB TO NC-1 DO           (* BUBBLE SORT ARRAY *)
( 40)   53        FOR J:=I+IB TO NC DO IF CA[J]^.COST < CA[I]^.COST THEN BEGIN
( 40)   54           P:=CA[J];       CA[J]:=CA[I];       CA[I]:=P;
( 40)   55           END;
( 40)   56        I:=MAXS+1;
( 40)   57        IF AQP.TOLER[IC]=TRUNC(AQP.TOLER[IC]) THEN(* CHECK AND COMPUTE TOLERANCES *)
( 40)   58        X:=AQP.TOLER[IC]
( 40)   59        ELSE X:=AQP.TOLER[IC]*(CA[NC]^.COST-CA[1]^.COST);
( 40)   60        IF IC<>AQP.NF THEN             (* IF NOT LAST FUNCTION, ADVANCE THE POINTER AS LONG AS *)
( 40)   61        WHILE (CA[MAXS]^.COST >= CA[I]^.COST-X) AND (I<=NC) DO I:=I+1;(* WE'RE WITHIN THE TOLERANCE *)
( 40)   62         (* RETURN ELEMENTS FROM I TO NC*)
( 40)   63
( 40)   64  2:    FOR J:=I TO NC DO BEGIN
( 40)   65           CA[J]^.NXTC:=AQP.FREEC;        (* RELEASE TRIMMED COMPLEXES *)
( 40)   66           AQP.FREEC:=CA[J];                 FREETAB(CA[J]^.TAB);(* INCLUDING ASSOCIATED TABLES *)
( 40)   67           END;
( 40)   68        NC:=I-1;       IB:=MAXS-1;
( 40)   69        WHILE(CA[MAXS]^.COST <= CA[IB]^.COST+X) AND (IB>0) DO IB:=IB-1;
( 40)   70        IB:=IB+1;       IC:=IC+1;
( 40)   71        IF IC<=AQP.NF THEN GOTO 1;     (* IF NOT DONE, DO IT AGAIN *)
( 40)   72  99:   NSTAR:=NIL;                    (* COMPUTE NEW STAR BY ADDING THESE ONTO HEAD OF NSTAR *)
( 40)   73        FOR I:=1 TO NC DO BEGIN
( 40)   74           CA[I]^.NXTC:=NSTAR;       NSTAR:=CA[I];
( 40)   75           END;
( 40)   76        END;
( 41)    1  *DECK COSTF
( 41)    2  FUNCTION COSTF
( 41)    3       (P              : CPTR;          (* EVALUATE AQ COSTS, P IS COMPLEX TO COST *)
( 41)    4        CT             : INTEGER): INTEGER;(* CT IS COST FUNCTION TO USE *)
( 41)    5  LABEL
```

```
( 41)    6          6;
( 41)    7    VAR
( 41)    8          I,J,K            : INTEGER;
( 41)    9          INSD             : BOOLEAN;
( 41)   10          G1               : GPTR;
( 41)   11          CTNEG            : BOOLEAN;
( 41)   12          Q                : CPTR;
( 41)   13    BEGIN  (*COSTF*)
( 41)   14
( 41)   15          IF CT<0 THEN                        (* CHECK AND REMEMBER THE SIGN OF COST FUNCTION INDEX *)
( 41)   16          CTNEG:=TRUE
( 41)   17          ELSE CTNEG:=FALSE;
( 41)   18          CT:=ABS(CT);                        (* THEN TAKE ABSOLUTE VALUE *)
( 41)   19          P^.COST:=0;                         (* INIT COST TO ZERO *)
( 41)   20          CASE CT OF
( 41)   21    3:       BEGIN                            (* COST TYPE 3, ACTUAL NUMBER OF RULES COVERED BY THIS CPX *)
( 41)   22                  G1:=GSET;                   (* GO THRU THE RULE BASE *)
( 41)   23                  K:=0;                       (* AND COUNT EM *)
( 41)   24                  WHILE G1<>NIL DO BEGIN
( 41)   25                     IF G1^.FP AND (ES IN G1^.ESET) THEN(* IF ITS IN F1 AND UNDER CONSIDERATION *)
( 41)   26                     BEGIN
( 41)   27                        J:=1;      K:=K+1;
( 41)   28                        IF GSUB^.MSEL<>NIL THEN(* IF THERE ARE META SELECTORS, CHECK THEM FIRST *)
( 41)   29                        FOR J:=1 TO MST.NMST DO GSUB^.MSEL^.CVAL[MST.PTR[J]]:=P^.CVAL[J];
( 41)   30                        J:=MST.NMST+1;        (* COPY MST VALUES TO GSUB *)
( 41)   31                        FOR I:=1 TO GSIZE DO (* THEN GO THRU THE NODEX OF THE GRAPH *)
( 41)   32                        IF (GSUB^.COUNT[I]=1) THEN  (* IF THIS NODE IS "IN" *)
( 41)   33                        BEGIN
( 41)   34                           GSUB^.VAL[I]:=P^.CVAL[J];   (* COPY NODE VALUE TO GSUB *)
( 41)   35                           J:=J+1;
( 41)   36                           END;              (* NOW WE HAVE A GRAPH OF THIS COMPLEX, LETS USE GRAPH *)
( 41)   37                        IF SUBG1(GSUB,G1,0,AQP.FREEC,TRUE,NIL) THEN(* MATCHER TO SEE IF WE COVER F1 *)
( 41)   38                        P^.COST:=P^.COST+1;  (* IF SO, ADD 1 TO THE COST *)
( 41)   39                        END;
( 41)   40                  G1:=G1^.NXTN;               (* NEXT RULE PLEASE ... *)
( 41)   41                  IF(MAXS>1)AND(K>AQP.CUTF1)THEN(* STOP WHEN WE HIT MAXSTAR AND CUTF1 *)
( 41)   42                  G1:=NIL;
( 41)   43                  END;(*WHILE G1<>NIL*)
( 41)   44
( 41)   45               END;(*CASE 1*)
( 41)   46
( 41)   47    2,4,7:  BEGIN                               (* 2: NUMBER OF NON * SELECTORS, 4: COSTS OF NON * SELECTORS *)
( 41)   48               FOR J:=1 TO AQP.NVAR DO     (* LOOK AT EACH VARIABLE *)
( 41)   49               IF ([0..MNVAL]-P^.CVAL[J])<>[] THEN(* IF NON * *)
( 41)   50               CASE CT OF
( 41)   51
( 41)   52    2:            P^.COST:=P^.COST+1;       (* COUNT IT, IF CT=2 *)
( 41)   53    4:            P^.COST:=P^.COST+S.VCOST[AQP.SLOC[J]]; (* ADD IN ITS COST IF CT=4 *)
( 41)   54    7:            BEGIN
( 41)   55                     IF AQT <> NIL THEN BEGIN
( 41)   56                        IF AQT^.CVAL[J]<>[] THEN P^.COST := P^.COST + 1;
( 41)   57                        END
( 41)   58                     ELSE P^.COST := 0;
( 41)   59                     END;(* CASE 7 *)
( 41)   60                  END;(* CASE *)
( 41)   61               END;(* CASE 2,4,7 *)
( 41)   62             (*CASE 2*)
( 41)   63
```

```
( 41)   64  1,5,6:   BEGIN
( 41)   65              CASE CT OF
( 41)   66  1:             BEGIN
( 41)   67                    Q:=E1;                (* COUNT NUMBER OF "NEW" EVENTS NEWLY COVERED--EXAMINE E1 LIST *)
( 41)   68                    INSD:=TRUE;           (* COMPLETE COVERS PLEASE *)
( 41)   69                    END;
( 41)   70  5:             BEGIN
( 41)   71                    Q:=F1;                (* EXAMINE ALL EVENTS COVERED--EXAMINE F1 LIST *)
( 41)   72                    INSD:=TRUE;           (* COMPLETE COVERS *)
( 41)   73                    END;
( 41)   74  6:             BEGIN
( 41)   75                    Q:=F2;                (* EXAMINE ALL FO EVENTS COVERED *)
( 41)   76                    INSD:=FALSE;          (* INTERSECTION IS ALL WE NEED *)
( 41)   77                    END
( 41)   78                 END;(*CASE STMT*)
( 41)   79
( 41)   80              WHILE Q<>NIL DO              (* INSPECT THE LIST SPECIFIED ABOVE *)
( 41)   81              BEGIN
( 41)   82                 IF ((CT=1)AND Q^.FQ)OR(CT IN [5,6]) THEN(* IF CT=1, IT MUST BE IN FQ SET, ELSE ANYTHING *)
( 41)   83                 FOR I:=1 TO AQP.NVAR DO (* EXAMINE EACH VARIABLE *)
( 41)   84                    IF INSD THEN
( 41)   85                       IF NOT(Q^.CVAL[I]<=P^.CVAL[I])THEN(* IF P COVERS Q, Q'S VALUES ARE A SUBSET OF P *)
( 41)   86                       GOTO 6             (* THUS, JUMP OUT IF IT DIDN'T COVER *)
( 41)   87                       ELSE
( 41)   88                    ELSE                  (* INSD = FALSE *)
( 41)   89                    IF NOT(Q^.CVAL[I]*P^.CVAL[I]<>[]) THEN    (* DO THEY INTERSECT? *)
( 41)   90                    GOTO 6;               (* IF THEY DON'T INTERSECT, JUMP OUT *)
( 41)   91                 P^.COST:=P^.COST+1;      (* WELL... COUNT THAT RULE *)
( 41)   92  6:             Q:=Q^.NXTC;              (* NEXT RULE PLEASE *)
( 41)   93                 END;
( 41)   94              END(*CASE 3*)
( 41)   95
( 41)   96           END;(*CASE STMT*)
( 41)   97
( 41)   98        IF CTNEG THEN                     (* FIX SIGN OF COST *)
( 41)   99        P^.COST:=-P^.COST;       COSTF:=P^.COST;(* AND RETURN AS VALUE OF FUNCTION *)
( 41)  100        END;
( 42)    1  *DECK CLEANCPX
( 42)    2  PROCEDURE CLEANCPX
( 42)    3       (F                : CPTR);(* PURPOSE:  TO CLEAN THE STRUCTURED VARIABLES OF A COMPLEX.
( 42)    4      DURING THE GROWING PROCESS, WE SIMPLY "OR" IN ALL OF THE
( 42)    5       VALUES.  THIS ROUTINE ATTEMPTS TO FIND A HIGHER LEVEL CONCEPT WHICH
( 42)    6       WILL GENERALIZE THE ENTIRE REFERENCE.
( 42)    7    *)
( 42)    8  VAR
( 42)    9        I,J               : INTEGER;(* TEMP INTEGERS *)
( 42)   10  BEGIN
( 42)   11        WITH F^ DO BEGIN
( 42)   12           FOR I:= 1 TO AQP.NVAR DO BEGIN
( 42)   13              IF (S.VTYPE[AQP.SLOC[I]]=3) AND (CVAL[I]<>[0..MNVAL]) THEN BEGIN
( 42)   14                 J:= 1;
( 42)   15                 WHILE (J<=DST.NELE) AND (NOT ((DST.PNO[J]=AQP.SLOC[I]) AND (CVAL[I]<=DST.PREM[J]))) DO J:=J+1;
( 42)   16                 IF J<=DST.NELE THEN CVAL[I]:=DST.CONS[J];
( 42)   17                 END;(* IF STRUCTURED *)
( 42)   18              END;(* FOR I *)
( 42)   19           END;(* WITH F *)
( 42)   20        END;(* CLEANCPX *)
( 43)    1  *DECK LQST2
```

```
( 43)    2  PROCEDURE LQST2
( 43)    3       (F              : CPTR;
( 43)    4        VAR F1         : CPTR);(* PURPOSE:
( 43)    5       LQST2 PERFORMS A LIMITED GENERALIZATION PROCESS ON THE SET F
( 43)    6       OF COMPLEXES WHICH IT IS GIVEN.  THIS GENERALIZATION CORRESPONDS TO
( 43)    7       A CHARACTERISTIC GENERALIZATION OF THE ORIGINAL SET OF RULES FROM
( 43)    8       WHICH F WAS BUILT (SEE AQSET BELOW).  EACH COMPLEX ON F HAS ASSOCIATED
( 43)    9       WITH IT A CPXTABLE (BUILT BY ADDC SEE ABOVE)  WHICH GIVES A LIST OF
( 43)   10       THE RULE NUMBERS OF THE RULES IN GSET FROM WHICH THAT PARTICULAR
( 43)   11       COMPLEX WAS BUILT.  LQST2 BUILDS A COVER WHICH COVERS (IN A QUASI
( 43)   12       OPTIMAL FASHION) THOSE RULES WITH AS SMALL A COVER AS POSSIBLE.
( 43)   13
( 43)   14       THE BASIC IDEA IS TO CHOOSE THE COMPLEX WHICH COVERS THE MOST RULES,
( 43)   15       ELIMINATE IT, AND ALL THE RULES WHICH IT COVERS FROM FURTHER
( 43)   16       CONSIDERATION, AND REPEAT UNTIL ALL RULES HAVE BEEN COVERED.
( 43)   17       THE FINAL COVER IS THE DISJUNCTION (UNION OF REFERENCES) OF THE
( 43)   18       SELECTED COMPLEXES.  IF THERE IS A TIE BETWEEN COMPLEXES AS TO WHICH
( 43)   19       COVERS THE MOST RULES, THE COMPLEX WHICH WILL ADD THE FEWEST ELEMENTS
( 43)   20       TO THE REFERENCE IS CHOSEN.  *)
( 43)   21  VAR
( 43)   22        C              : CPTR;          (* POINTER TO AN ACTIVE COMPLEX.  IF NON NIL, IMPLIES THAT *)
( 43)   23                                        (* SOME RULE REMAINS TO BE COVERED *)
( 43)   24
( 43)   25   (*              I N D U C E - 1  --  LQST2  --  MAXCOUNT
( 43)   26   *)
( 43)   27  *CALL MAXCOUNT
( 43)   28   (*1
( 43)   29                 I N D U C E - 1  --  LQST2  --  ADDEDCOST
( 43)   30   *)
( 43)   31  *CALL ADDEDCOST
( 43)   32   (*1
( 43)   33                 I N D U C E - 1  --  LQST2  --  MAXCPXS
( 43)   34   *)
( 43)   35  *CALL MAXCPXS
( 43)   36   (*1
( 43)   37                 I N D U C E - 1  --  LQST2 (MAIN CODE)
( 43)   38   *)
( 43)   39  BEGIN
( 43)   40        NEWC(F1);          (* GET NEW CPX *)
( 43)   41        FOR I:=1 TO AQP.NVAR DO F1^.CVAL[I]:=[];(* CLEAN F1 COMPLEX *)
( 43)   42        REPEAT MAXCPXS(F,F1,C) UNTIL C=NIL;
( 43)   43        END;(*LQST2*)
( 44)    1  *DECK MAXCOUNT
( 44)    2  FUNCTION MAXCOUNT
( 44)    3       (F              : CPTR) : INTEGER;(* PURPOSE:  TO RETURN THE MAXIMUM COVCOUNT OF THE CPXTABLES ON THE
( 44)    4      CPX LIST F.
( 44)    5   *)
( 44)    6  VAR
( 44)    7        MAX            : INTEGER;       (* CURRENT MAXIMUM VALUE, EVENTUALLY RETURNED AS RESULT *)
( 44)    8        C              : CPTR;          (* TEMP POINTER TO CURRENT COMPLEX TO CONSIDER *)
( 44)    9  BEGIN
( 44)   10        MAX := 0;      C := F;          (* LOOP THRU F LIST *)
( 44)   11        WHILE C<>NIL DO BEGIN           (* AND INSPECT EACH TAB *)
( 44)   12           WITH C^.TAB^ DO BEGIN
( 44)   13              IF COVCOUNT>MAX THEN MAX := COVCOUNT;
( 44)   14              END;                     (* WITH *)
( 44)   15           C:=C^.NXTC;                 (* NEXT CPX *)
( 44)   16           END;(* WHILE *)
```

```
 44)   17        MAXCOUNT := MAX;                    (* RETURN VALUE *)
 44)   18        END;(* MAXCOUNT *)
 45)    1   *DECK ADDEDCOST
 45)    2   FUNCTION ADDEDCOST
 45)    3        (C,F             : CPTR) : INTEGER;(* PURPOSE
 45)    4        TO COMPUTE THE ADDITIONAL NUMBER OF VALUES IN REFERENCES OF F
 45)    5        THAT WOULD BE INTRODUCED BY ADDING C INTO THE COVERING COMPLEX F.
 45)    6        FOR NOMINAL VARIABLES, THIS IS JUST THE NUMBER OF NEW VALUES IN THE
 45)    7        REFERENCE.  FOR INTERVAL VARIABLES, THIS INCLUDES ANY NEW VALUES
 45)    8        WHICH MUST BE ADDED TO FILL OUT THE INTERVAL BETWEEN THE SMALLEST
 45)    9        AND LARGEST VALUES OF THE NEW REFERENCE.  AND FOR STRUCTURED VARIABLES,
 45)   10        WE WANT TO KNOW HOW MANY TERMINAL LEAVES WILL BE ADDED WHEN WE
 45)   11        GENERALIZE THE NEW REFERENCE.  WE COMPARE THIS NUMBER TO THE
 45)   12        CURRENT NUMBER.
 45)   13        THE COST RETURNED IS THE SUM OF THESE COSTS.
 45)   14    *)
 45)   15   VAR
 45)   16        COST             : INTEGER;          (* TEMP COPY OF COST *)
 45)   17        I,J,K            : INTEGER;          (* TEMP WORKING INDECES *)
 45)   18        LOWER, UPPER     : INTEGER;          (* LOWER AND UPPER LIMITS OF THE INTERVAL REFERENCE *)
 45)   19        V,V1             : VALTP;            (* TEMP VERSION OF THE MERGED REFERENCE *)
 45)   20   BEGIN
 45)   21        COST:=0;
 45)   22        IF 7 IN TRACE THEN BEGIN
 45)   23           WRITELN(OUTPUT,'ADDED COST OF ADDING');      PCPX(C);      WRITELN(OUTPUT,'TO');      PCPX(F);
 45)   24           END;(*IF*)
 45)   25        FOR I := 1 TO AQP.NVAR DO BEGIN  (* I POINTS TO INDIVIDUAL VARIABLE *)
 45)   26           V := F^.CVAL[I] + C^.CVAL[I]; (* MERGE THE 2 REFERENCES *)
 45)   27           IF (V<>F^.CVAL[I]) AND (V<>[0..MNVAL]) THEN BEGIN(* ONLY DO THIS IF THERE IS A CHANGE *)
 45)   28              CASE S.VTYPE[AQP.SLOC[I]] OF
 45)   29   1:            (*NOMINAL*)
 45)   30                 COST := COST + CARD(V-F^.CVAL[I]);
 45)   31
 45)   32   2:            (*INTERVAL*)
 45)   33                 BEGIN
 45)   34                    LOWER:=0;             (* FIND SMALLEST VALUE IN REFERENCE *)
 45)   35                    WHILE (NOT (LOWER IN V)) AND (LOWER<=MNVAL) DO LOWER:=LOWER+1;
 45)   36                    UPPER:=MNVAL;         (* FIND LARGEST VALUE IN REFERENCE *)
 45)   37                    WHILE (NOT (UPPER IN V)) AND (UPPER>=0) DO UPPER:=UPPER-1;
 45)   38                    IF (UPPER-LOWER)>=0 THEN COST := COST + (UPPER-LOWER+1) - CARD(F^.CVAL[I]);
 45)   39                    END;(* CASE 2 *)
 45)   40
 45)   41   3:            (*STRUCTURED*)
 45)   42                 BEGIN
 45)   43                    IF CARD(V)=1 THEN COST:=COST+1(* IF SINGLE ELEMENT, QUIT *)
 45)   44                    ELSE BEGIN
 45)   45                       J:=1;                 (* FIND FIRST DST RULE WHICH GENERALIZES V *)
 45)   46                       WHILE (J<=DST.NELE) AND (NOT ((DST.PNO[J]=AQP.SLOC[I]) AND (V<=DST.PREM[J]))) DO J:=J+1;
 45)   47
 45)   48                       IF (J<=DST.NELE) THEN BEGIN
 45)   49                          V:=DST.PREM[J]+DST.CONS[J];  (* GET ENTIRE PREMISE AND CONS *)
 45)   50                          COST := COST + CARD(V)-CARD(F^.CVAL[I]);
 45)   51                          END;(* IF J<=NELE *)
 45)   52                       END;(* ELSE CARD<>1 *)
 45)   53                    END;(*CASE 3*)
 45)   54                 END;(* CASE *)
 45)   55              END;(* IF *)
 45)   56           END;(* FOR *)
```

```
 45)   57        ADDEDCOST := COST;
 45)   58        IF 7 IN TRACE THEN WRITELN(OUTPUT,COST:3);
 45)   59        END;(*ADDEDCOST*)
 46)    1   *DECK MAXCPXS
 46)    2   PROCEDURE MAXCPXS
 46)    3        (F, F1           : CPTR;
 46)    4        VAR C            : CPTR);(* PURPOSE:  TO PERFORM ONE COVERING ITERATION.
 46)    5       1) FIRST DETERMINE THE MAXIMUM NUMBER OF RULES COVERED BY A SINGLE
 46)    6          COMPLEX.
 46)    7       2) DETERMINE WHICH OF THE COMPLEXES WHICH COVER THIS MAX NUMBER OF
 46)    8          RULES HAS THE SMALLEST ADDED COST RELATIVE TO THE CURRENT COVER.
 46)    9       3) CHOOSE THE SMALLEST AND MERGE IT INTO THE CURRENT COVER.
 46)   10       4) ADJUST THE REFERENCES OF EACH VARIABLE IN THE COMPLEX IN LIGHT
 46)   11          OF THE MERGE.
 46)   12       5) UPDATE THE CPXTABLES OF EACH CPX TO REMOVE THOSE RULES WHICH HAVE
 46)   13          NOW BEEN COVERED.
 46)   14
 46)   15       F: IS THE SET OF COMPLEXES TO COVER
 46)   16       F1: IS THE CURRENT COVER
 46)   17       C: IS A POINTER WHICH RETURNS THE LAST CPX WHICH STILL HAS A NONZERO
 46)   18          COVCOUNT.  IT INDICATES THAT THERE IS STILL WORK TO DO.
 46)   19    *)
 46)   20   VAR
 46)   21        MAX              : INTEGER;(* INDICATES MAX NUMBER OF RULES COVERED BY A
 46)   22                                SINGLE COMPLEX IN F *)
 46)   23        MIN              : INTEGER;(* USED TO COMPUTE MINIMUM ADDED COST *)
 46)   24        I,J,K            : INTEGER;(* TEMP INTEGERS *)
 46)   25        C1               : CPTR;(* TEMP CPTR *)
 46)   26        MINC             : CPTR;(* POINTER TO COMPLEX WITH MINIMUM ADDED COST *)
 46)   27        TMPCVAL          : VALTP;          (* TEMP REFERENCE SET TO AVOID COMPILER LIMIT *)
 46)   28        UPPER,LOWER      : INTEGER;(* USED TO ADJUST REFERENCES OF INTERVAL VARIABLES *)
 46)   29        T                : CPXTABPTR;(* USED TO STORE CPXTAB DURING STEP 5 *)
 46)   30   BEGIN
 46)   31
 46)   32      (* STEP 1 :  DETERMINE MAX COVER *)
 46)   33
 46)   34        MAX := MAXCOUNT (F);
 46)   35        IF 7 IN TRACE THEN WRITELN(OUTPUT,'MAXCOUNT IS ',MAX:3);
 46)   36
 46)   37      (* STEP 2 :  DETERMINE RULE WITH MIN ADDED COST *)
 46)   38
 46)   39        MINC := NIL;                      (* INIT POINTER *)
 46)   40        MIN := MAXINT;                    (* INIT MINIMUM TO + INFINITY *)
 46)   41        C1:=F;                            (* LOOP THRU F *)
 46)   42        WHILE C1<>NIL DO BEGIN
 46)   43           IF C1^.TAB^.COVCOUNT=MAX THEN BEGIN
 46)   44              I:=ADDEDCOST(C1,F1);        (* COMPUTE COST OF ADDING C1 *)
 46)   45              IF I<MIN THEN BEGIN
 46)   46                 MIN:=I;                  (* UPDATE CURRENT MINIMUM *)
 46)   47                 MINC:=C1;                (* REMEMBER THE MINIMUM CPX *)
 46)   48                 END;(* IF I<MIN*)
 46)   49              END;(* IF =MAX *)
 46)   50           C1:=C1^.NXTC;                  (* DO NEXT CPX *)
 46)   51           END;(*WHILE<>NIL*)
 46)   52        IF 7 IN TRACE THEN BEGIN
 46)   53           WRITELN(OUTPUT,'SELECTED CPX IS ');      PCPX(MINC);
 46)   54           END;
 46)   55
```

```
( 44)   17        MAXCOUNT := MAX;                    (* RETURN VALUE *)
( 44)   18        END;(* MAXCOUNT *)
( 45)    1  *DECK ADDEDCOST
( 45)    2  FUNCTION ADDEDCOST
( 45)    3       (C,F              : CPTR) : INTEGER;(* PURPOSE
( 45)    4        TO COMPUTE THE ADDITIONAL NUMBER OF VALUES IN REFERENCES OF F
( 45)    5        THAT WOULD BE INTRODUCED BY ADDING C INTO THE COVERING COMPLEX F.
( 45)    6        FOR NOMINAL VARIABLES, THIS IS JUST THE NUMBER OF NEW VALUES IN THE
( 45)    7        REFERENCE.  FOR INTERVAL VARIABLES, THIS INCLUDES ANY NEW VALUES
( 45)    8        WHICH MUST BE ADDED TO FILL OUT THE INTERVAL BETWEEN THE SMALLEST
( 45)    9        AND LARGEST VALUES OF THE NEW REFERENCE.  AND FOR STRUCTURED VARIABLES,
( 45)   10        WE WANT TO KNOW HOW MANY TERMINAL LEAVES WILL BE ADDED WHEN WE
( 45)   11        GENERALIZE THE NEW REFERENCE.  WE COMPARE THIS NUMBER TO THE
( 45)   12        CURRENT NUMBER.
( 45)   13        THE COST RETURNED IS THE SUM OF THESE COSTS.
( 45)   14    *)
( 45)   15  VAR
( 45)   16        COST              : INTEGER;          (* TEMP COPY OF COST *)
( 45)   17        I,J,K             : INTEGER;          (* TEMP WORKING INDECES *)
( 45)   18        LOWER, UPPER      : INTEGER;          (* LOWER AND UPPER LIMITS OF THE INTERVAL REFERENCE *)
( 45)   19        V,V1              : VALTP;            (* TEMP VERSION OF THE MERGED REFERENCE *)
( 45)   20  BEGIN
( 45)   21        COST:=0;
( 45)   22        IF 7 IN TRACE THEN BEGIN
( 45)   23           WRITELN(OUTPUT,'ADDED COST OF ADDING');      PCPX(C);      WRITELN(OUTPUT,'TO');      PCPX(F);
( 45)   24           END;(*IF*)
( 45)   25        FOR I := 1 TO AQP.NVAR DO BEGIN  (* I POINTS TO INDIVIDUAL VARIABLE *)
( 45)   26           V := F^.CVAL[I] + C^.CVAL[I]; (* MERGE THE 2 REFERENCES *)
( 45)   27           IF (V<>F^.CVAL[I]) AND (V<>[0..MNVAL]) THEN BEGIN(* ONLY DO THIS IF THERE IS A CHANGE *)
( 45)   28              CASE S.VTYPE[AQP.SLOC[I]] OF
( 45)   29  1:             (*NOMINAL*)
( 45)   30                 COST := COST + CARD(V-F^.CVAL[I]);
( 45)   31
( 45)   32  2:             (*INTERVAL*)
( 45)   33                 BEGIN
( 45)   34                    LOWER:=0;              (* FIND SMALLEST VALUE IN REFERENCE *)
( 45)   35                    WHILE (NOT (LOWER IN V)) AND (LOWER<=MNVAL) DO LOWER:=LOWER+1;
( 45)   36                    UPPER:=MNVAL;          (* FIND LARGEST VALUE IN REFERENCE *)
( 45)   37                    WHILE (NOT (UPPER IN V)) AND (UPPER>=0) DO UPPER:=UPPER-1;
( 45)   38                    IF (UPPER-LOWER)>=0 THEN COST := COST + (UPPER-LOWER+1) - CARD(F^.CVAL[I]);
( 45)   39                    END;(* CASE 2 *)
( 45)   40
( 45)   41  3:             (*STRUCTURED*)
( 45)   42                 BEGIN
( 45)   43                    IF CARD(V)=1 THEN COST:=COST+1(* IF SINGLE ELEMENT, QUIT *)
( 45)   44                    ELSE BEGIN
( 45)   45                       J:=1;                (* FIND FIRST DST RULE WHICH GENERALIZES V *)
( 45)   46                       WHILE (J<=DST.NELE) AND (NOT ((DST.PNO[J]=AQP.SLOC[I]) AND (V<=DST.PREM[J]))) DO J:=J+1;
( 45)   47
( 45)   48                       IF (J<=DST.NELE) THEN BEGIN
( 45)   49                          V:=DST.PREM[J]+DST.CONS[J];  (* GET ENTIRE PREMISE AND CONS *)
( 45)   50                          COST := COST + CARD(V)-CARD(F^.CVAL[I]);
( 45)   51                          END;(* IF J<=NELE *)
( 45)   52                       END;(* ELSE CARD<>1 *)
( 45)   53                    END;(*CASE 3*)
( 45)   54                 END;(* CASE *)
( 45)   55              END;(* IF *)
( 45)   56           END;(* FOR *)
```

```
( 45)   57        ADDEDCOST := COST;
( 45)   58        IF 7 IN TRACE THEN WRITELN(OUTPUT,COST:3);
( 45)   59        END;(*ADDEDCOST*)
( 46)    1  *DECK MAXCPXS
( 46)    2  PROCEDURE MAXCPXS
( 46)    3       (F, F1          : CPTR;
( 46)    4        VAR C          : CPTR);(* PURPOSE:  TO PERFORM ONE COVERING ITERATION.
( 46)    5      1) FIRST DETERMINE THE MAXIMUM NUMBER OF RULES COVERED BY A SINGLE
( 46)    6         COMPLEX.
( 46)    7      2) DETERMINE WHICH OF THE COMPLEXES WHICH COVER THIS MAX NUMBER OF
( 46)    8         RULES HAS THE SMALLEST ADDED COST RELATIVE TO THE CURRENT COVER.
( 46)    9      3) CHOOSE THE SMALLEST AND MERGE IT INTO THE CURRENT COVER.
( 46)   10      4) ADJUST THE REFERENCES OF EACH VARIABLE IN THE COMPLEX IN LIGHT
( 46)   11         OF THE MERGE.
( 46)   12      5) UPDATE THE CPXTABLES OF EACH CPX TO REMOVE THOSE RULES WHICH HAVE
( 46)   13         NOW BEEN COVERED.
( 46)   14
( 46)   15      F: IS THE SET OF COMPLEXES TO COVER
( 46)   16      F1: IS THE CURRENT COVER
( 46)   17      C: IS A POINTER WHICH RETURNS THE LAST CPX WHICH STILL HAS A NONZERO
( 46)   18         COVCOUNT.  IT INDICATES THAT THERE IS STILL WORK TO DO.
( 46)   19   *)
( 46)   20  VAR
( 46)   21        MAX             : INTEGER;(* INDICATES MAX NUMBER OF RULES COVERED BY A
( 46)   22                                SINGLE COMPLEX IN F *)
( 46)   23        MIN             : INTEGER;(* USED TO COMPUTE MINIMUM ADDED COST *)
( 46)   24        I,J,K           : INTEGER;(* TEMP INTEGERS *)
( 46)   25        C1              : CPTR;(* TEMP CPTR *)
( 46)   26        MINC            : CPTR;(* POINTER TO COMPLEX WITH MINIMUM ADDED COST *)
( 46)   27        TMPCVAL         : VALTP;          (* TEMP REFERENCE SET TO AVOID COMPILER LIMIT *)
( 46)   28        UPPER,LOWER     : INTEGER;(* USED TO ADJUST REFERENCES OF INTERVAL VARIABLES *)
( 46)   29        T               : CPXTABPTR;(* USED TO STORE CPXTAB DURING STEP 5 *)
( 46)   30  BEGIN
( 46)   31
( 46)   32     (* STEP 1 :  DETERMINE MAX COVER *)
( 46)   33
( 46)   34        MAX := MAXCOUNT (F);
( 46)   35        IF 7 IN TRACE THEN WRITELN(OUTPUT,'MAXCOUNT IS ',MAX:3);
( 46)   36
( 46)   37     (* STEP 2 :  DETERMINE RULE WITH MIN ADDED COST *)
( 46)   38
( 46)   39        MINC := NIL;                      (* INIT POINTER *)
( 46)   40        MIN := MAXINT;                    (* INIT MINIMUM TO + INFINITY *)
( 46)   41        C1:=F;                            (* LOOP THRU F *)
( 46)   42        WHILE C1<>NIL DO BEGIN
( 46)   43           IF C1^.TAB^.COVCOUNT=MAX THEN BEGIN
( 46)   44              I:=ADDEDCOST(C1,F1);        (* COMPUTE COST OF ADDING C1 *)
( 46)   45              IF I<MIN THEN BEGIN
( 46)   46                 MIN:=I;                  (* UPDATE CURRENT MINIMUM *)
( 46)   47                 MINC:=C1;                (* REMEMBER THE MINIMUM CPX *)
( 46)   48                 END;(* IF I<MIN*)
( 46)   49              END;(* IF =MAX *)
( 46)   50           C1:=C1^.NXTC;                  (* DO NEXT CPX *)
( 46)   51           END;(*WHILE<>NIL*)
( 46)   52        IF 7 IN TRACE THEN BEGIN
( 46)   53           WRITELN(OUTPUT,'SELECTED CPX IS ');      PCPX(MINC);
( 46)   54           END;
( 46)   55
```

```
( 44)   17        MAXCOUNT := MAX;                    (* RETURN VALUE *)
( 44)   18        END;(* MAXCOUNT *)
( 45)    1  *DECK ADDEDCOST
( 45)    2  FUNCTION ADDEDCOST
( 45)    3       (C,F              : CPTR) : INTEGER;(* PURPOSE
( 45)    4        TO COMPUTE THE ADDITIONAL NUMBER OF VALUES IN REFERENCES OF F
( 45)    5        THAT WOULD BE INTRODUCED BY ADDING C INTO THE COVERING COMPLEX F.
( 45)    6        FOR NOMINAL VARIABLES, THIS IS JUST THE NUMBER OF NEW VALUES IN THE
( 45)    7        REFERENCE.  FOR INTERVAL VARIABLES, THIS INCLUDES ANY NEW VALUES
( 45)    8        WHICH MUST BE ADDED TO FILL OUT THE INTERVAL BETWEEN THE SMALLEST
( 45)    9        AND LARGEST VALUES OF THE NEW REFERENCE.  AND FOR STRUCTURED VARIABLES,
( 45)   10        WE WANT TO KNOW HOW MANY TERMINAL LEAVES WILL BE ADDED WHEN WE
( 45)   11        GENERALIZE THE NEW REFERENCE.  WE COMPARE THIS NUMBER TO THE
( 45)   12        CURRENT NUMBER.
( 45)   13        THE COST RETURNED IS THE SUM OF THESE COSTS.
( 45)   14    *)
( 45)   15  VAR
( 45)   16        COST              : INTEGER;          (* TEMP COPY OF COST *)
( 45)   17        I,J,K             : INTEGER;          (* TEMP WORKING INDECES *)
( 45)   18        LOWER, UPPER      : INTEGER;          (* LOWER AND UPPER LIMITS OF THE INTERVAL REFERENCE *)
( 45)   19        V,V1              : VALTP;            (* TEMP VERSION OF THE MERGED REFERENCE *)
( 45)   20  BEGIN
( 45)   21        COST:=0;
( 45)   22        IF 7 IN TRACE THEN BEGIN
( 45)   23           WRITELN(OUTPUT,'ADDED COST OF ADDING');      PCPX(C);      WRITELN(OUTPUT,'TO');      PCPX(F);
( 45)   24           END;(*IF*)
( 45)   25        FOR I := 1 TO AQP.NVAR DO BEGIN  (* I POINTS TO INDIVIDUAL VARIABLE *)
( 45)   26           V := F^.CVAL[I] + C^.CVAL[I]; (* MERGE THE 2 REFERENCES *)
( 45)   27           IF (V<>F^.CVAL[I]) AND (V<>[0..MNVAL]) THEN BEGIN(* ONLY DO THIS IF THERE IS A CHANGE *)
( 45)   28              CASE S.VTYPE[AQP.SLOC[I]] OF
( 45)   29  1:             (*NOMINAL*)
( 45)   30                 COST := COST + CARD(V-F^.CVAL[I]);
( 45)   31
( 45)   32  2:             (*INTERVAL*)
( 45)   33                 BEGIN
( 45)   34                    LOWER:=0;              (* FIND SMALLEST VALUE IN REFERENCE *)
( 45)   35                    WHILE (NOT (LOWER IN V)) AND (LOWER<=MNVAL) DO LOWER:=LOWER+1;
( 45)   36                    UPPER:=MNVAL;          (* FIND LARGEST VALUE IN REFERENCE *)
( 45)   37                    WHILE (NOT (UPPER IN V)) AND (UPPER>=0) DO UPPER:=UPPER-1;
( 45)   38                    IF (UPPER-LOWER)>=0 THEN COST := COST + (UPPER-LOWER+1) - CARD(F^.CVAL[I]);
( 45)   39                    END;(* CASE 2 *)
( 45)   40
( 45)   41  3:             (*STRUCTURED*)
( 45)   42                 BEGIN
( 45)   43                    IF CARD(V)=1 THEN COST:=COST+1(* IF SINGLE ELEMENT, QUIT *)
( 45)   44                    ELSE BEGIN
( 45)   45                       J:=1;               (* FIND FIRST DST RULE WHICH GENERALIZES V *)
( 45)   46                       WHILE (J<=DST.NELE) AND (NOT ((DST.PNO[J]=AQP.SLOC[I]) AND (V<=DST.PREM[J]))) DO J:=J+1;
( 45)   47
( 45)   48                       IF (J<=DST.NELE) THEN BEGIN
( 45)   49                          V:=DST.PREM[J]+DST.CONS[J];  (* GET ENTIRE PREMISE AND CONS *)
( 45)   50                          COST := COST + CARD(V)-CARD(F^.CVAL[I]);
( 45)   51                          END;(* IF J<=NELE *)
( 45)   52                       END;(* ELSE CARD<>1 *)
( 45)   53                    END;(*CASE 3*)
( 45)   54                 END;(* CASE *)
( 45)   55              END;(* IF *)
( 45)   56           END;(* FOR *)
```

```
 45)   57        ADDEDCOST := COST;
 45)   58        IF 7 IN TRACE THEN WRITELN(OUTPUT,COST:3);
 45)   59        END;(*ADDEDCOST*)
 46)    1   *DECK MAXCPXS
 46)    2   PROCEDURE MAXCPXS
 46)    3        (F, F1           : CPTR;
 46)    4         VAR C           : CPTR);(* PURPOSE:  TO PERFORM ONE COVERING ITERATION.
 46)    5       1) FIRST DETERMINE THE MAXIMUM NUMBER OF RULES COVERED BY A SINGLE
 46)    6          COMPLEX.
 46)    7       2) DETERMINE WHICH OF THE COMPLEXES WHICH COVER THIS MAX NUMBER OF
 46)    8          RULES HAS THE SMALLEST ADDED COST RELATIVE TO THE CURRENT COVER.
 46)    9       3) CHOOSE THE SMALLEST AND MERGE IT INTO THE CURRENT COVER.
 46)   10       4) ADJUST THE REFERENCES OF EACH VARIABLE IN THE COMPLEX IN LIGHT
 46)   11          OF THE MERGE.
 46)   12       5) UPDATE THE CPXTABLES OF EACH CPX TO REMOVE THOSE RULES WHICH HAVE
 46)   13          NOW BEEN COVERED.
 46)   14
 46)   15       F: IS THE SET OF COMPLEXES TO COVER
 46)   16       F1: IS THE CURRENT COVER
 46)   17       C: IS A POINTER WHICH RETURNS THE LAST CPX WHICH STILL HAS A NONZERO
 46)   18          COVCOUNT.  IT INDICATES THAT THERE IS STILL WORK TO DO.
 46)   19    *)
 46)   20   VAR
 46)   21         MAX             : INTEGER;(* INDICATES MAX NUMBER OF RULES COVERED BY A
 46)   22                                SINGLE COMPLEX IN F *)
 46)   23         MIN             : INTEGER;(* USED TO COMPUTE MINIMUM ADDED COST *)
 46)   24         I,J,K           : INTEGER;(* TEMP INTEGERS *)
 46)   25         C1              : CPTR;(* TEMP CPTR *)
 46)   26         MINC            : CPTR;(* POINTER TO COMPLEX WITH MINIMUM ADDED COST *)
 46)   27         TMPCVAL         : VALTP;          (* TEMP REFERENCE SET TO AVOID COMPILER LIMIT *)
 46)   28         UPPER,LOWER     : INTEGER;(* USED TO ADJUST REFERENCES OF INTERVAL VARIABLES *)
 46)   29         T               : CPXTABPTR;(* USED TO STORE CPXTAB DURING STEP 5 *)
 46)   30   BEGIN
 46)   31
 46)   32      (* STEP 1 :  DETERMINE MAX COVER *)
 46)   33
 46)   34         MAX := MAXCOUNT (F);
 46)   35         IF 7 IN TRACE THEN WRITELN(OUTPUT,'MAXCOUNT IS ',MAX:3);
 46)   36
 46)   37      (* STEP 2 :  DETERMINE RULE WITH MIN ADDED COST *)
 46)   38
 46)   39         MINC := NIL;                      (* INIT POINTER *)
 46)   40         MIN := MAXINT;                    (* INIT MINIMUM TO + INFINITY *)
 46)   41         C1:=F;                            (* LOOP THRU F *)
 46)   42         WHILE C1<>NIL DO BEGIN
 46)   43            IF C1^.TAB^.COVCOUNT=MAX THEN BEGIN
 46)   44               I:=ADDEDCOST(C1,F1);        (* COMPUTE COST OF ADDING C1 *)
 46)   45               IF I<MIN THEN BEGIN
 46)   46                  MIN:=I;                  (* UPDATE CURRENT MINIMUM *)
 46)   47                  MINC:=C1;                (* REMEMBER THE MINIMUM CPX *)
 46)   48                  END;(* IF I<MIN*)
 46)   49               END;(* IF =MAX *)
 46)   50            C1:=C1^.NXTC;                  (* DO NEXT CPX *)
 46)   51            END;(*WHILE<>NIL*)
 46)   52         IF 7 IN TRACE THEN BEGIN
 46)   53            WRITELN(OUTPUT,'SELECTED CPX IS ');      PCPX(MINC);
 46)   54            END;
 46)   55
```

```
( 46)   56     (* STEP 3+4: MERGE THE TWO COMPLEXES  *)
( 46)   57
( 46)   58        FOR I:=1 TO AQP.NVAR DO BEGIN
( 46)   59           TMPCVAL:=F1^.CVAL[I];          (* GET AROUND COMPILER LIMITS *)
( 46)   60           TMPCVAL:=TMPCVAL + MINC^.CVAL[I];(* MERGE VALUES *)
( 46)   61           F1^.CVAL[I]:=TMPCVAL;
( 46)   62           END;(* FOR I *)
( 46)   63        IF 7 IN TRACE THEN BEGIN
( 46)   64           WRITELN(OUTPUT,'COVER AFTER MERGE IS:');      PCPX(F1);
( 46)   65           END;
( 46)   66
( 46)   67     (* STEP 5 :  UPDATE CPXTABLES TO REMOVE RULES THAT ARE NOW COVERED *)
( 46)   68
( 46)   69        C:=NIL;                            (* NO ACTIVE CPX YET *)
( 46)   70        NEWCPXTAB(T);                      (* GET A CPXTAB AS TEMP *)
( 46)   71        T^:=MINC^.TAB^;                    (* COPY MINC CPXTAB INTO T *)
( 46)   72        C1:=F;                             (* LOOP THRU F AGAIN *)
( 46)   73        IF 7 IN TRACE THEN WRITELN(OUTPUT,'SET OF CPXS IS NOW');
( 46)   74        WHILE C1<>NIL DO BEGIN
( 46)   75           K:=0;                           (* K IS USED AS "NEW" INDEX INTO COVLIST *)
( 46)   76           FOR J:=1 TO C1^.TAB^.COVCOUNT DO BEGIN
( 46)   77              I:=1;                        (* COPY RULE NUMBERS NOT FOUND IN MINC TAB *)
( 46)   78              WHILE (I<=T^.COVCOUNT) AND (C1^.TAB^.COVLIST[J]<>T^.COVLIST[I]) DO I:=I+1;
( 46)   79              IF (I>T^.COVCOUNT) THEN BEGIN
( 46)   80                 K:=K+1;                   (* WE HAVE AN ELEMENT NOT IN MINC.TAB *)
( 46)   81                 C1^.TAB^.COVLIST[K]:=C1^.TAB^.COVLIST[J];
( 46)   82                 C:=C1;                    (* REMEMBER THIS CPX *)
( 46)   83                 END;(*IF I>MINC COVCOUNT*)
( 46)   84              END;(*FOR J*)
( 46)   85           C1^.TAB^.COVCOUNT:=K;          (* UPDATE THE COV COUNT *)
( 46)   86           IF 7 IN TRACE THEN PCPX(C1);   (* FOR DEBUGGING *)
( 46)   87           C1:=C1^.NXTC;                  (* NEXT CPX PLEASE *)
( 46)   88           END;(* WHILE C1<>NIL*)
( 46)   89        FREETAB(T);                        (* FREE TEMP CPX TAB *)
( 46)   90        END;(*MAXCPXS *)
( 47)    1  *DECK AQSET
( 47)    2  PROCEDURE AQSET
( 47)    3       (GSET            : GPTR;           (* GSET : RULE BASE *)
( 47)    4        ES              : INTEGER;        (* ES : EVENT SET TO COVER *)
( 47)    5        GSUB            : GPTR);          (* GSUB : CURRENT RULE TO EXTEND *)
( 47)    6  LABEL
( 47)    7        1,
( 47)    8        3,
( 47)    9        4,
( 47)   10        99;                               (* UUUGGGHHH *)
( 47)   11  VAR
( 47)   12        G,G1            : GPTR;
( 47)   13        F,F1,F2         : CPTR;
( 47)   14        I,J,K,L         : INTEGER;
( 47)   15        DONE            : BOOLEAN;
( 47)   16  BEGIN  (* SET UP CLASS BEING COVERED *)
( 47)   17
( 47)   18        G:=GSET;
( 47)   19        WITH GSUB^ DO FOR I:=1 TO GSIZE DO(* ELIMINATE DUMMY VARIABLES WHEN THEY ONLY APPEAR *)
( 47)   20        IF ORDIRR[I] AND (NOT VBL[I]) THEN BEGIN(* AS DUMMIES OF ONE SELECTOR *)
( 47)   21           J:=1;
( 47)   22           WHILE LNK[I,J]<>0 DO BEGIN
( 47)   23              IF LNK[LNK[I,J],2]=0 THEN BEGIN
```

```
( 47)   24                LNK[LNK[I,J],1]:=0;        (* DISCONNNECT DUMMY COMPLETELY *)
( 47)   25                COUNT[LNK[I,J]]:=0;        (* RESET ITS COUNT *)
( 47)   26                LNK[I,J]:=GSIZE;          (* POINT ORIGINAL AT GSIZE SO I'LL EXTRACT IT BELOW *)
( 47)   27                END;
( 47)   28             J:=J+1;
( 47)   29             END;
( 47)   30          J:=1;       K:=1;
( 47)   31          WHILE LNK[I,J]<>0 DO          (* COPY THE LINKS SKIPPING THE ONES POINTED AT GSIZE *)
( 47)   32          BEGIN
( 47)   33             IF LNK[I,J]<>GSIZE THEN BEGIN
( 47)   34                LNK[I,K]:=LNK[I,J];       K:=K+1;
( 47)   35                END;
( 47)   36             J:=J+1;
( 47)   37             END;
( 47)   38          LNK[I,K]:=0;                  (* MAKE SURE MY LINK LIST TERMINATES IN 0 *)
( 47)   39          END;
( 47)   40       F1:=NIL;                      (* BUILD COMPLEX LISTS, F: FIRST SET OF CPX'S ON LIST, INCLUDING METASELECTORS *)
( 47)   41       F2:=NIL;(* F1: REMAINDER OF F1 SET OF CPX'S F IS PUT AT HEAD OF F1 *)
( 47)   42       F:=NIL;                       (* F2: SET OF CPX'S FOR FO SET *)
( 47)   43       WHILE G<>NIL DO               (* FIND 1ST GRAPH WITH AN ISOMORPHISM TO THIS RULE *)
( 47)   44       BEGIN
( 47)   45          IF (ES IN G^.ESET)AND(G^.FP) THEN (* IF IN F1 AND UNDER CONSIDERATION *)
( 47)   46          BEGIN
( 47)   47             IF SUBG1(GSUB,G,2,F,TRUE,NIL) THEN  (* IF GSUB IS SUBGRAPH OF G *)
( 47)   48             ;
( 47)   49             GOTO 4;                   (* BUILD ALL ISOMORPHISM COMPLEXES AND PUT IN F *)
( 47)   50             END;
( 47)   51          G:=G^.NXTN;
( 47)   52          END;(* WHILE *)
( 47)   53
( 47)   54        (* CLEAR ALL VAL FIELDS OF GSUB *)
( 47)   55
( 47)   56  4:    FOR I:=1 TO GSIZE DO GSUB^.VAL[I]:=[0..MNVAL];(* SET ALL MSEL VALUE TO * *)
( 47)   57   (*  IF GSUB^.MSEL=NIL THEN *)
( 47)   58       NEWC(GSUB^.MSEL);             (* GET A NEW MSEL CPX-- *)
( 47)   59       FOR I:=1 TO GSIZE DO GSUB^.MSEL^.CVAL[I]:=[0..MNVAL];
( 47)   60       G:=G^.NXTN; (*DEBUG*)
( 47)   61       WHILE G<>NIL DO               (* FOR EACH REMAINING GRAPH, *)
( 47)   62       BEGIN                         (* FIGURE ALL ISOMORPHISMS *)
( 47)   63          IF (ES IN G^.ESET) AND G^.FP THEN
( 47)   64             IF SUBG1(GSUB,G,2,F1,TRUE,NIL) THEN(* AND PUT THEM IN F1 COMPLEX LIST *)
( 47)   65             ;
( 47)   66          G:=G^.NXTN;
( 47)   67          END;
( 47)   68       F^.NXTC:=F1;      F1:=F;          (* PUT F AT HEAD OF F1 *)
( 47)   69       G:=GSET;                      (* PLACE ALL INTERSECTING CPX'S IN FO *)
( 47)   70       WHILE G<>NIL DO               (* ONTO THE F2 LIST *)
( 47)   71       BEGIN
( 47)   72          IF NOT(ES IN G^.ESET) THEN    (* IS G IN FO? *)
( 47)   73             IF SUBG1(GSUB,G,2,F2,FALSE,NIL) THEN(* AND G INTERSECTS GSUB *)
( 47)   74             ;
( 47)   75          G:=G^.NXTN;                   (* PUT ON F2 *)
( 47)   76          END;(*WHILE*)
( 47)   77
( 47)   78       IF 4 IN TRACE THEN BEGIN
( 47)   79          EXPLN(4);      WRITELN(OUTPUT,'THE C-FORMULA STRUCTURE IS:');      PGRAPH(GSUB,S);
( 47)   80          WRITELN(OUTPUT,'THERE ARE ',AQP.NVAR:3,' VL1 TYPE VARIABLES X1,', 'X2,...,X', AQP.NVAR:2);
( 47)   81          WRITELN(OUTPUT,'VARIABLES ARE ASSOCIATED WITH NODES IN THE C-FORMULA', ' AS FOLLOWS:');
```

```
( 47)   82          WRITELN(OUTPUT);
( 47)   83          WRITELN(OUTPUT,'                NODE','                    VARIABLE'); J:=MST.NMST+1;
( 47)   84          FOR I:=1 TO GSIZE DO IF GSUB^.COUNT[I]=1 THEN BEGIN
( 47)   85             WRITE(OUTPUT,'                ');       K:=1;
( 47)   86             WHILE S.NAME[ABS(AQP.SLOC[J]),K]<>' ' DO BEGIN
( 47)   87                WRITE(OUTPUT,S.NAME[ABS(AQP.SLOC[J]),K]);       K:=K+1;
( 47)   88                END;
( 47)   89             IF GSUB^.VBL[I] THEN BEGIN
( 47)   90                IF GSUB^.DUMNUM[I]>9 THEN WRITE(OUTPUT,GSUB^.DUMNUM[I]:2) ELSE WRITE(OUTPUT,GSUB^.DUMNUM[I]:1);
( 47)   91                K:=K+1;
( 47)   92                END;
( 47)   93             FOR L:=K TO 20 DO WRITE(OUTPUT,' ');
( 47)   94             IF J>9 THEN WRITELN(OUTPUT,'X',J:2) ELSE WRITELN(OUTPUT,'X',J:1);
( 47)   95             J:=J+1;
( 47)   96             END;
( 47)   97          WRITELN(OUTPUT,'AQ IS APPLIED TO THE FOLLOWING INPUT CPXS/EVENTS');
( 47)   98          WRITELN(OUTPUT,'                ','                SET 1');
( 47)   99          F:=F1;
( 47)  100          WHILE F<>NIL DO BEGIN
( 47)  101             PCPX(F);       F:=F^.NXTC;
( 47)  102             END;
( 47)  103          WRITELN(OUTPUT,'                ','                SET 2');       F:=F2;
( 47)  104          WHILE F<>NIL DO BEGIN
( 47)  105             PCPX(F);       F:=F^.NXTC;
( 47)  106             END;
( 47)  107          END;(*TRACE OF 4*)
( 47)  108
( 47)  109  3:    CASE PRM.DESCTYPE OF
( 47)  110
( 47)  111  DISCRIMINANT:
( 47)  112          F:=AQ(GSUB,FALSE,F1,F2);              (*PERFORM AQ ALG. GET BEST EXSENSION OF F1 AGAINST F2 *)
( 47)  113  CHARACTERISTIC:
( 47)  114          LQST2(F1,F);                    (* PERFORM MINIMUM COVER OF F1, PUT IN F *)
( 47)  115          END;(* CASE *)
( 47)  116       IF F=NIL THEN                      (* IF NONE FOUND...*)
( 47)  117       GOTO 99;                           (* QUIT... *)
( 47)  118       IF PRM.DESCTYPE=CHARACTERISTIC THEN CLEANCPX(F);(* SIMPLIFY THE STRUCTURED VARIABLES *)
( 47)  119       IF 5 IN TRACE THEN BEGIN
( 47)  120          EXPLN(5);      WRITELN(OUTPUT,'THE RESULTING COMPLEX FROM THIS PASS IS:');      PCPX(F);
( 47)  121          END;(*TRACE 5*)
( 47)  122
( 47)  123       ;(* TRANSLATE COVER INTO GRAPH *)
( 47)  124       J:=0;
( 47)  125       FOR J:=1 TO MST.NMST DO GSUB^.MSEL^.CVAL[MST.PTR[J]]:=F^.CVAL[J];(* COPY META SELECTOR VALUES BACK INTO GRAPH *)
( 47)  126       J:=MST.NMST;
( 47)  127       FOR I:=1 TO GSIZE DO               (* COPY REST OF VALUES INTO GRAPH *)
( 47)  128       IF GSUB^.COUNT[I]=1 THEN BEGIN
( 47)  129          J:=J+1;       GSUB^.VAL[I]:=F^.CVAL[J];
( 47)  130          END;
( 47)  131       FREECPX(F);                        (* PUT F ON FREE LIST *)
( 47)  132  99:   FREECPX(F1);                       (* RELEASE F1 LIST *)
( 47)  133       FREECPX(F2);                       (* RELEASE F2 LIST *)
( 47)  134       END;  (*AQSET*)
( 48)    1  *DECK ENTERD
( 48)    2  PROCEDURE ENTERD;                        (* ENTER DOMAIN STRUCTURES *)
( 48)    3  VAR
( 48)    4       I              : INTEGER;
( 48)    5  BEGIN
```

```
( 48)    6        NEWG(G);                          (* GET A NEW G STRUCTURE *)
( 48)    7        VLINT(G,ERR,ES,VLRULE,NIL);       (* READ IN A RULE AND PUT IT INTO GSTRUCTURE *)
( 48)    8        WITH DST DO BEGIN                 (* COPY RELEVANT INFO INTO DST *)
( 48)    9           NELE:=NELE+1;                  (* COUNT # OF ELEMENTS IN DST *)
( 48)   10           IF NELE>NDES THEN              (* IF TOO MANY...*)
( 48)   11           WRITELN(OUTPUT,'DOMAIN STRUC TABLE OVFL (NDES =',NDES:4,')');
( 48)   12           PREM[NELE]:=G^.VAL[1];         (* SET OF VALUES FOR NODE1 = PREMISE *)
( 48)   13           CONS[NELE]:=G^.VAL[2];         (* SET OF VALUES FOR NODE2 = CONSEQUENCE *)
( 48)   14           PNO[NELE]:=G^.PNO[1];          (* DESCRIPTOR NUMBER *)
( 48)   15           FOR I:=1 TO NELE DO IF PNO[I]=PNO[NELE] THEN(* NOW SEARCH DST, IF OTHER DEFNS USE SAME *)
( 48)   16              IF CONS[I]<=PREM[NELE] THEN  (* DESCRIPTOR AND THEIR CONS IS IN OUR PREM *)
( 48)   17              PREM[NELE]:=PREM[NELE]+PREM[I];    (* THEN ADD THEIR PREMISE TO OURS *)
( 48)   18           END;(*WITH*)
( 48)   19
( 48)   20        G^.NXTN:=FREEG;                   (* RELEASE THE GSTRUC *)
( 48)   21        FREEG:=G;
( 48)   22        END; (* ENTERD *)
( 49)    1     *DECK VL1
( 49)    2     PROCEDURE VL1;                       (* GO INTO VL1 MODE OF OPERATION *)
( 49)    3     LABEL
( 49)    4           1,
( 49)    5           2,
( 49)    6           3;
( 49)    7     VAR
( 49)    8           F1,F2,F,P,Q,AQE: CPTR;
( 49)    9           I,J,K,ES       : INTEGER;
( 49)   10           AGNST          : VALTP;
( 49)   11     BEGIN
( 49)   12           AQE:=NIL;      RESET(VL1EVE);      F:=NIL;
( 49)   13           WITH S DO WITH AQP DO BEGIN(*SETUP NELT, NAME,PNO*)
( 49)   14
( 49)   15              WRITELN(OUTPUT,'HOW MANY VARIABLES');      ILINE;      READ(IFILE,NVAR);      S.NELT:=AQP.NVAR;
( 49)   16              FOR I:=1 TO NVAR DO BEGIN
( 49)   17                 NAME[I]:='          ';      VTYPE[I]:=1;      S.NAME[I,1]:='X';
( 49)   18                 IF I>9 THEN BEGIN
( 49)   19                    S.NAME[I,2]:=CHR((I DIV 10)+ORD('0'));      S.NAME[I,3]:=CHR((I MOD 10)+ORD('0'));
( 49)   20                    END
( 49)   21                 ELSE S.NAME[I,2]:=CHR(I+ORD('0'));
( 49)   22                 PNO[I]:=I;      SLOC[I]:=I;      DPNO[I]:=I;
( 49)   23                 END;
( 49)   24              WHILE NOT EOF(VL1EVE) DO BEGIN
( 49)   25                 NEWC(Q);      Q^.NXTC:=AQE;      AQE:=Q;      READ(VL1EVE,I);
( 49)   26                 IF I>=0 THEN Q^.CVAL[NVAR+1]:=[I] ELSE Q^.CVAL[NVAR+1]:=[0..MNVAL];
( 49)   27                 FOR I:=1 TO NVAR DO BEGIN
( 49)   28                    READ(VL1EVE,J);
( 49)   29                    IF J IN [0..MNVAL] THEN Q^.CVAL[I]:=[J] ELSE Q^.CVAL[I]:=[0..MNVAL];
( 49)   30                    IF J<MVAL[I] THEN MVAL[I]:=J;
( 49)   31                    IF J>EVAL[I] THEN EVAL[I]:=J;
( 49)   32                    IF J>NVAL[I] THEN NVAL[I]:=J;
( 49)   33                    END;
( 49)   34                 READLN(VL1EVE);
( 49)   35                 END;
( 49)   36              END;(*READ EVENTS*)
( 49)   37
( 49)   38     2:       WRITELN(OUTPUT,'ENTER P TO CHANGE PARAMETERS');
( 49)   39              WRITELN(OUTPUT,'      C TO COVER EVENTS');      WRITELN(OUTPUT,'      E TO ENTER DOMAIN STRUCTURE');
( 49)   40              WRITELN(OUTPUT,'      Q TO RETURN TO MAIN LEVEL');
( 49)   41              WRITELN(OUTPUT);                                ILINE;
```

```
( 49)   42         GETCHRR(CHRR);
( 49)   43         IF CHRR IN ['C','Q','E','P'] THEN WITH AQP DO WITH S DO CASE CHRR OF
( 49)   44            'P': ENTERP;
( 49)   45            'E': ENTERD;
( 49)   46            'C': BEGIN
( 49)   47               WRITELN(OUTPUT,'ENTER DECISION NUMBER OF SET TO BE COVERED');      ILINE;      READ(IFILE,ES);
( 49)   48               WRITELN(OUTPUT,'AGAINST WHICH SETS, ENTER NUMBERS', 'FOR THESE SETS OR ENTER -1 TO COVER AGAINST ALL');
( 49)   49               ILINE;                                                              AGNST:=[];
( 49)   50               WHILE NOT EOLN(IFILE) DO BEGIN
( 49)   51                  READ(IFILE,I);
( 49)   52                  IF I=-1 THEN BEGIN
( 49)   53                     AGNST:=[0..MNVAL]-[ES];
( 49)   54                     GOTO 3;
( 49)   55                     END;
( 49)   56                  AGNST:=AGNST+[I];
( 49)   57                  END;
( 49)   58  3:             F1:=NIL;
( 49)   59                 F2:=NIL;      Q:=AQE;      AQE:=NIL;
( 49)   60                 WHILE Q<>NIL DO BEGIN
( 49)   61                    P:=Q^.NXTC;
( 49)   62                    IF ES IN Q^.CVAL[NVAR+1] THEN BEGIN
( 49)   63                       Q^.NXTC:=F1;      F1:=Q;
( 49)   64                       END
( 49)   65                    ELSE IF Q^.CVAL[NVAR+1] <= AGNST THEN BEGIN
( 49)   66                       Q^.NXTC:=F2;      F2:=Q;
( 49)   67                       END
( 49)   68                    ELSE BEGIN
( 49)   69                       Q^.NXTC:=AQE;      AQE:=Q;
( 49)   70                       END;
( 49)   71                    Q:=P;
( 49)   72                    END;
( 49)   73                 IF (F1<>NIL) THEN BEGIN
( 49)   74                    F:=AQ(G,TRUE,F1,F2);      (* CALL AQ IN VL1 MODE *)
( 49)   75                    WRITELN(OUTPUT,'OUTPUT COMPLEXES FOR SET',ES:3);
( 49)   76                    Q:=F;
( 49)   77                    WHILE Q<>NIL DO BEGIN
( 49)   78                       P:=Q;      PCPX(Q);      Q:=Q^.NXTC;
( 49)   79                       END;
( 49)   80                    P^.NXTC:=FREEC;      FREEC:=F;
( 49)   81                    END;
( 49)   82                 IF F1<>NIL THEN BEGIN
( 49)   83                    P:=F1;
( 49)   84                    WHILE P^.NXTC<>NIL DO P:=P^.NXTC;
( 49)   85                    P^.NXTC:=AQE;      AQE:=F1;
( 49)   86                    END;
( 49)   87                 IF F2 <> NIL THEN BEGIN
( 49)   88                    P:=F2;
( 49)   89                    WHILE P^.NXTC<>NIL DO P:=P^.NXTC;
( 49)   90                    P^.NXTC:=AQE;      AQE:=F2;
( 49)   91                    END;
( 49)   92                 END;    (*CASE C*)
( 49)   93
( 49)   94            'Q': GOTO 1
( 49)   95            END;(*CASE STMT*)
( 49)   96
( 49)   97         GOTO 2;
( 49)   98  1:     FREECPX(AQE);
( 49)   99         END; (* VL1 *)
```

```
( 50)    1  *DECK NEWGP
( 50)    2  PROCEDURE NEWGP
( 50)    3       (ALTER          : INTEGER;          (* ALTER: NUMBER OF ALTERNATIVES TO GENERATE *)
( 50)    4        GO,G1          : GPTR;             (* GO: CURRENT GENERALIZATION, G1: RULE TO COVER, 1:1 BY INDEX *)
( 50)    5        VAR SLST       : GPTR);            (* SLST: STAR LIST TO ADD THE NEW GROUP TO *)
( 50)    6  LABEL
( 50)    7        1,
( 50)    8        2;
( 50)    9  VAR
( 50)   10        I,J,K,L,M,CPTR : INTEGER;
( 50)   11        CANDID         : ARRAY[1..GSIZE] OF INTEGER;(* CANDIDATE SELECTOR ARRAY *)
( 50)   12        G              : GPTR;
( 50)   13  BEGIN  (*NEWGP*)
( 50)   14
( 50)   15         (*GENERATE A LIST OF ALL SELECTORS WHICH MAY BE CONNECTED
( 50)   16           TO THE GRAPH. GO IS OLD GRAPH,    G1 IS
( 50)   17           EVENT WHICH IS BEING COVERED COUNT=1,NDE FROM OLD
( 50)   18           GRAPH COUNT=2 NODE IS VARIABLE CONNECTED TO OLD
( 50)   19           GRAPH COUNT=3 NODE IS NEW SELECTOR *)
( 50)   20
( 50)   21        FOR I:=1 TO GSIZE DO               (* EXAMINE G1 GRAPH *)
( 50)   22        IF GO^.COUNT[I]<>0 THEN
( 50)   23           IF GO^.PNO[I]<0 THEN            (* IS IT AN EQUIVALENCE TYPE SELECTOR? *)
( 50)   24           BEGIN                           (* PUT ALL DUMMIES OF EQUIVALENCE TYPE PREDICATES INTO GRAPH *)
( 50)   25              J:=1;
( 50)   26              WHILE G1^.LNK[I,J]<>0 DO BEGIN
( 50)   27                 IF GO^.COUNT[G1^.LNK[I,J]]=0 THEN GO^.COUNT[G1^.LNK[I,J]]:=2;
( 50)   28                 J:=J+1;
( 50)   29                 END;                     (* SET THEIR COUNTS TO 2 *)
( 50)   30           END;
( 50)   31        CPTR:=0;
( 50)   32        FOR I:=1 TO GSIZE DO               (* EXAMINE GO, IF THIS NODE *)
( 50)   33        IF GO^.VBL[I] AND (GO^.COUNT[I]>0) THEN  (* IS A VARIABLE WITH A NON-ZERO COUNT *)
( 50)   34        BEGIN                              (* SET ALL FUNCTIONS AND PREDICATES CONNECTED TO IT *)
( 50)   35           J:=1;                           (* TO "NEW", COUNT=3 *)
( 50)   36           WHILE G1^.LNK[I,J]<>0 DO BEGIN
( 50)   37              IF GO^.COUNT[G1^.LNK[I,J]]=0 THEN GO^.COUNT[G1^.LNK[I,J]]:=3;
( 50)   38              J:=J+1;
( 50)   39              END;
( 50)   40           END;
( 50)   41        FOR I:=1 TO GSIZE DO               (* EXAMINE GO AGAIN *)
( 50)   42        BEGIN                              (* IF ITS A NEW NODE *)
( 50)   43           IF(GO^.COUNT[I]=3) THEN BEGIN
( 50)   44              CPTR:=CPTR+1;               (* COUNT IT AND PUT IT IN CANDID ARRAY *)
( 50)   45              CANDID[CPTR]:=I;
( 50)   46              END;
( 50)   47           IF GO^.COUNT[I]<>1 THEN        (* IF NOT OLD, SET COUNT TO ZERO--REMOVE THOSE *)
( 50)   48           GO^.COUNT[I]:=0;               (* EQUIVALENCE-TYPE DUMMIES AGAIN *)
( 50)   49           END;(*SORT CANDID ARRAY IF ALTER < CPTR*)
( 50)   50
( 50)   51        IF (ALTER<>0)AND(ALTER<CPTR) THEN FOR I:=1 TO CPTR-1 DO(* SORT TO TOP LOW COST DESCRIPTORS, OR *)
( 50)   52        FOR J:=I+1 TO CPTR DO              (* DESCRIPTORS WITH FEW DUMMIES *)
( 50)   53           IF (S.VCOST[GO^.PNO[CANDID[I]]]> S.VCOST [ GO^.PNO[CANDID[J]]]) OR
( 50)   54              (S.VCOST[GO^.PNO [CANDID[I]]]=S.VCOST[GO^.PNO[CANDID [J]]]) AND
( 50)   55              (S.NARG[GO^.PNO[CANDID[I]]]>S.NARG[GO^.PNO[CANDID[J]]])
( 50)   56              THEN BEGIN(* SWAP THESE ENTRIES IF HIGHER COST, OR MORE DUMMIES *)
( 50)   57              L:=CANDID[I];      CANDID[I]:=CANDID[J];      CANDID[J]:=L;
( 50)   58              END;(*FORM NEW GRAPH FOR EACH ALTERNATIVE SELECTOR*)
```

```
( 50)   59
( 50)   60        M:=0;
( 50)   61        FOR I:=1 TO CPTR DO                (* FOR EACH ALTERNATIVE SELECTOR *)
( 50)   62        BEGIN
( 50)   63           NEWG(G);       G^:=GO^;        (* GET A NEW GRAPH AND COPY GO INFORMATION TO IT *)
( 50)   64           G^.COUNT[CANDID[I]]:=1;
( 50)   65           G^.RNO:=CRULENO-1;
( 50)   66           J:=1;
( 50)   67           IF G^.PNO[CANDID[I]]>0 THEN    (* FOR NORMAL (NON EQUIVALENCE ) SELECTORS *)
( 50)   68           WHILE G1^.LNK[CANDID[I],J]<>0 DO BEGIN(* SET CONNECTED DUMMY VARIABLES TO COUNT=1 *)
( 50)   69              G^.COUNT[G1^.LNK[CANDID[I],J]]:=1;      J:=J+1;
( 50)   70              END;
( 50)   71           FOR J:=1 TO GSIZE DO IF (G1^.LNK[J,1]<>0)AND(G^.COUNT[J]<>0) THEN BEGIN(* IF IT IS IN GRAPH AND COUNT > 0 *)
( 50)   72              K:=1;       L:=1;
( 50)   73              WHILE G1^.LNK[J,K]<>0 DO     (* FOR EACH ACTIVE LINK *)
( 50)   74              BEGIN                        (* COPY LINKS CONNECTED TO ACTIVE NODEX *)
( 50)   75                 IF G^.COUNT[G1^.LNK[J,K]]=1 THEN BEGIN
( 50)   76                    G^.LNK[J,L]:=G1^.LNK[J,K];      L:=L+1;
( 50)   77                    END;
( 50)   78                 K:=K+1;
( 50)   79                 END;    (*IF G1*)
( 50)   80
( 50)   81              G^.LNK[J,L]:=0;              (* TERMINATE LINK LIST IN 0 *)
( 50)   82              IF (G^.PNO[J]<0)AND(L=2) THEN    (* RETURN EQUIVALENCE PREDICATES WITH ONLY *)
( 50)   83              BEGIN                        (* A SINGLE DUMMY ARGUMENT *)
( 50)   84                 G^.NXTN:=FREEG;       FREEG:=G;
( 50)   85                 GOTO 1;
( 50)   86                 END;
( 50)   87              END;(*FOR J*)
( 50)   88
( 50)   89           G^.NXTN:=SLST;                  (* ADD THIS GRAPH TO THE SLST--NEW STAR *)
( 50)   90           SLST:=G;                     M:=M+1;
( 50)   91           IF (ALTER<>0)AND(M>=ALTER) THEN (* DO IT UNTIL WE HAVE ENOUGH *)
( 50)   92           GOTO 2;
( 50)   93  1:       ;
( 50)   94           END;(*FOR I*)
( 50)   95
( 50)   96  2:
( 50)   97        END; (* NEWGP *)
( 51)    1  *DECK PGRAPH
( 51)    2  PROCEDURE PGRAPH;                        (* ARGS ARE G^ AND S^ *)
( 51)    3  LABEL
( 51)    4        1;
( 51)    5  VAR
( 51)    6        I,J,K,L,M,NSEL : INTEGER;
( 51)    7        DSIZE,VALSIZE  : INTEGER;
( 51)    8        COMPVAL        : VALTP;
( 51)    9        MTEMP          : -SYMSZE .. SYMSZE;
( 51)   10        TD1            : INTEGER;(*TEMP USED IN MS PRINT ROUTINE*)
( 51)   11   (*1
( 51)   12                 I N D U C E - 1  --  PGRAPH  --  WRITEVAL
( 51)   13   *)
( 51)   14  *CALL WRITEVAL
( 51)   15   (*1
( 51)   16                 I N D U C E - 1  --  PGRAPH (MAIN CODE)
( 51)   17   *)
( 51)   18
( 51)   19  BEGIN
```

```
( 51)   20        J:=0;       WRITE(OUTPUT,'RULE  # ',G^.RNO:5);(* PRINT THE RNO *)
( 51)   21        IF G^.ESET<>[] THEN WRITE(OUTPUT,' EVENT SETS:');(* AND THE EVENT SETS IT BELONGS TO *)
( 51)   22        FOR I:=0 TO MNVAL DO IF I IN G^.ESET THEN WRITE(OUTPUT,I:3);
( 51)   23        WRITE(OUTPUT,' COSTS(');              (* AND THE COST FUNCTIONS *)
( 51)   24        FOR I:=1 TO PRM.NF DO                 (* THESE WILL BE 0 IF IT HASNT BEEN EVALUATED YET *)
( 51)   25        WRITE(OUTPUT,PRM.CSTF[I]:2);          WRITE(OUTPUT,')');
( 51)   26        FOR I:=1 TO PRM.NF DO
( 51)   27           IF G^.COST[ABS(PRM.CSTF[I])]<>-100 THEN WRITE(OUTPUT,G^.COST[ABS(PRM.CSTF[I])]:5) ELSE WRITE(OUTPUT,0:5);
( 51)   28        WRITELN(OUTPUT);       NSEL:=0;       (* COUNT OF NUMBER OF SELECTORS PRINTED *)
( 51)   29        IF PRULE THEN                         (* SHOULD WE PRINT THE RULES THEMSELVES?*)
( 51)   30        WITH G^ DO BEGIN
( 51)   31           FOR I:=1 TO GSIZE DO               (* ASSIGN SUBSCRIPTS TO THE DUMMY VARIABLES *)
( 51)   32           IF VBL[I] AND (G^.LNK[I,1]<>0) THEN     (* IF ITS A DUMMY WITH A LINK *)
( 51)   33           BEGIN
( 51)   34              J:=J+1;       DUMNUM[I]:=J; (* ASSIGN THE NEXT NUMBER TO IT *)
( 51)   35              END;
( 51)   36           FOR I:=1 TO GSIZE DO IF (LNK[I,1]<>0) THEN(* IF IT HAS A LINK *)
( 51)   37              IF (NOT VBL[I])OR VBL[I] AND(VAL[I]<>[0..MNVAL])THEN    (* AND ITS NOT A DUMMY OR *)
( 51)   38              BEGIN                          (* ITS A DUMMY WITH A NON * REFERENCE *)
( 51)   39                 NSEL:=NSEL+1;               (* PRINT IT....COUNT THE SELECTORS *)
( 51)   40                 WRITE(OUTPUT,'[');
( 51)   41                 L:=ABS(PNO[I]);
( 51)   42                 FOR J:=1 TO 10 DO           (* PRINT THE NAME *)
( 51)   43                 IF S.NAME[L,J]<>' ' THEN WRITE(OUTPUT,S.NAME[L,J]);
( 51)   44                 DSIZE := S.NVAL[L] - S.MVAL[L]+1;      VALSIZE := CARD(VAL[I]);
( 51)   45                 IF NOT VBL[I] THEN          (* IF ITS NOT A VBL, PRINT ITS ARGS *)
( 51)   46                 BEGIN
( 51)   47                    WRITE(OUTPUT,'(');       J:=1;
( 51)   48                    WHILE LNK[I,J]<>0 DO BEGIN
( 51)   49                       M:=LNK[I,J];
( 51)   50                       FOR K:=1 TO 10 DO        (* PRINT NAME OF ARGUMENT *)
( 51)   51                       IF S.NAME[PNO[M],K]<>' ' THEN WRITE(OUTPUT,S.NAME[PNO[M],K]);
( 51)   52                       IF DUMNUM[M]>9 THEN       (* PRINT SUBSCRIPT OF ARGUMENT *)
( 51)   53                       WRITE(OUTPUT,DUMNUM[M]:2)
( 51)   54                       ELSE WRITE(OUTPUT,DUMNUM[M]:1);
( 51)   55                       J:=J+1;                (* INCREMENT LINK PTR *)
( 51)   56                       IF LNK[I,J]<>0 THEN       (* IF NEXT LINK IS NON-TRIVIAL *)
( 51)   57                          IF PNO[I]>0 THEN WRITE(OUTPUT,',')(* DETECT THE =SAME CASE IF PNO<0 *)
( 51)   58                          ELSE WRITE(OUTPUT,'.')
( 51)   59                       ELSE IF PNO[I]<0 THEN      (* IF =SAME CASE, PRINT )=  *)
( 51)   60                       WRITE(OUTPUT,')=')
( 51)   61                       ELSE IF S.MVAL[PNO[I]]=S.NVAL[PNO[I]] THEN WRITE(OUTPUT,')')(* IF ITS A PREDICATE, PRINT ) *)
( 51)   62                       ELSE BEGIN
( 51)   63                          MTEMP := PNO[I];
( 51)   64                          IF (DSIZE < 2*VALSIZE) AND (S.VTYPE[MTEMP] <> 3) AND (VAL[I] <> [0..MNVAL]) THEN BEGIN
( 51)   65                             COMPVAL := [S.MVAL[MTEMP] .. S.NVAL[MTEMP]]-VAL[I];      WRITE(OUTPUT,')#');
( 51)   66                             END
( 51)   67                          ELSE WRITE(OUTPUT,')=');
( 51)   68                          END;
( 51)   69                       END;          (*WHILE*)
( 51)   70
( 51)   71                    END       (*NOT VBL*)
( 51)   72
( 51)   73                 ELSE IF DUMNUM[I]>9 THEN(* PRINT DUMNUM FOR THE [VAR I=K] CASE *)
( 51)   74                 WRITE(OUTPUT,DUMNUM[I]:2)
( 51)   75                 ELSE WRITE(OUTPUT,DUMNUM[I]:1);
( 51)   76                 IF PNO[I]>0 THEN        (* IF NOT =SAME CASE *)
( 51)   77                    IF S.NVAL[PNO[I]]<>S.MVAL[PNO[I]] THEN    (* AND NOT A PREDICATE *)
```

```
( 51)   78                      IF VAL[I]=[0..MNVAL] THEN   (* CHECK FOR =* CASE *)
( 51)   79                      WRITE(OUTPUT,' * ')
( 51)   80                      ELSE BEGIN
( 51)   81                         IF S.VTYPE[PNO[I]]=3 THEN    (* IF STRUCTURED VARIABLE *)
( 51)   82                         FOR M:=S.EVAL[PNO[I]] DOWNTO S.NVAL[PNO[I]]+1 DO
( 51)   83                            IF M IN VAL[I] THEN     (* FIND THE HIGHEST ELEMENT IN REFERENCE AND PRINT IT *)
( 51)   84                            BEGIN
( 51)   85                               WRITEVAL(M,PNO[I]);
( 51)   86                               GOTO 1;
( 51)   87                               END;
( 51)   88                         IF DSIZE < 2*VALSIZE THEN FOR M:=S.MVAL[PNO[I]] TO S.NVAL[PNO[I]] DO BEGIN
( 51)   89                            IF M IN COMPVAL THEN WRITEVAL(M,PNO[I])
( 51)   90                            END
( 51)   91                         ELSE FOR M:=S.MVAL[PNO[I]] TO S.NVAL[PNO[I]] DO IF M IN VAL[I] THEN WRITEVAL(M,PNO[I]);
( 51)   92  1:                      ;
( 51)   93                         END;
( 51)   94                IF PNO[I]<0 THEN        (* FOR THE =SAME, PRINT SAME *)
( 51)   95                WRITE(OUTPUT,'SAME');      WRITE(OUTPUT,']');(* CLOSE THE SELECTOR *)
( 51)   96                IF NSEL>=4 THEN BEGIN   (* PRINT ONLY 4 SELECTORS TO A LINE *)
( 51)   97                   NSEL:=0;      WRITELN(OUTPUT);      WRITE(OUTPUT,'     ');(* INDENT NEXT LINE *)
( 51)   98                   END;
( 51)   99                END;(*LNK<>0*)
( 51)  100
( 51)  101             END;(*WITH*)
( 51)  102
( 51)  103          WRITELN(OUTPUT);                (* TERMINATE THAT LINE *)
( 51)  104
( 51)  105          (*  PRINT METASELECTORS FOR THIS RULE *)
( 51)  106
( 51)  107          IF G^.MSEL<>NIL THEN            (* IF THERE ARE METASELECTORS FOR THIS RULE *)
( 51)  108          FOR I:=1 TO MST.NMST DO         (* FOR EACH METASELECTOR *)
( 51)  109             IF G^.MSEL^.CVAL[MST.PTR[I]]<>[0..MNVAL] THEN (* IF ITS NON-TRIVIAL *)
( 51)  110             BEGIN                        (* PRINT IT *)
( 51)  111                CASE MST.SYMPTR[MST.PTR[I]] OF
( 51)  112  1:                WRITE(OUTPUT,'[ALL ');          (* IT IS A FORALL TYPE*)
( 51)  113  2:                WRITE(OUTPUT,'[# ');            (* IT IS A #PT TYPE*)
( 51)  114                   END;
( 51)  115                IF MST.VARPTR[MST.PTR[I]]>0 THEN BEGIN  (* IF VARPTR IS OK *)
( 51)  116                   FOR TD1:= 1 TO 10 DO             (* WRITE THE VARIABLE NAME *)
( 51)  117                   IF S.NAME[MST.VARPTR[MST.PTR[I]],TD1] <> ' ' THEN WRITE(OUTPUT,S.NAME[MST.VARPTR[MST.PTR[I]],TD1]);
( 51)  118                   WRITE(OUTPUT,'S ');              (* MAKE IT PLURAL *)
( 51)  119                   END; (* IF VARPTR IS OK *)
( 51)  120                FOR TD1:= 1 TO 10 DO             (* WRITE THE FUNCTION NAME *)
( 51)  121                IF S.NAME[MST.PNO[MST.PTR[I]],TD1] <> ' ' THEN WRITE(OUTPUT,S.NAME[MST.PNO[MST.PTR[I]],TD1]);
( 51)  122                WRITEVAL(MST.VAL[MST.PTR[I]], MST.PNO[MST.PTR[I]]);
( 51)  123                IF MST.SYMPTR[MST.PTR[I]] = 2    (* IF THIS IS A #PT TYPE *)
( 51)  124                THEN BEGIN                       (* WRITE OUT THE VALUES OF MS*)
( 51)  125                   WRITE(OUTPUT,'=');
( 51)  126                   FOR J:=S.MVAL[MST.SYMPTR[MST.PTR[I]]] TO S.NVAL[MST.SYMPTR[MST.PTR[I]]] DO
( 51)  127                      IF J IN G^.MSEL^.CVAL[MST.PTR[I]] THEN WRITE(OUTPUT,J:2);(* PRINT THE VALUES OF THE SELECTOR *)
( 51)  128                   END;(*IF # PT*)
( 51)  129                WRITE(OUTPUT,']');          (* CLOSE THE SELECTOR *)
( 51)  130                END;
( 51)  131          WRITELN(OUTPUT);      WRITELN(OUTPUT);
( 51)  132          END; (* PGRAPH *)
( 52)    1  *DECK WRITEVAL
( 52)    2  PROCEDURE WRITEVAL
( 52)    3       (M,TD1          : INTEGER);         (* PRINT OUT NUMERIC OR SYMBOLIC VALUE *)
```

```
( 52)   (*  M IS SCALAR VALUE TO PRINT.  TD1 IS PNO OF CORRESPONDING DESCRIPTOR *)
( 52)  VAR
( 52)         TD2,TD3          : INTEGER;
( 52)  BEGIN
( 52)         TD2:=-1;            (* COMPUTE VALUE USING TD2 *)
( 52)         WHILE (S.DPNO[TD1]<>0) AND (TD2<>M) DO BEGIN
( 52)            TD1:=S.DPNO[TD1];(* FOLLOW SYMBOL LINKS THRU DPNO *)
( 52)            TD2:=TD2+1;     (* TD2 CONTAINS VALUE OF CORRESPONDING SYMBOL *)
( 52)            END;(* WHILE *)
( 52)         WRITE(OUTPUT,' ');
( 52)         IF TD2=M THEN BEGIN(* WE FOUND IT *)
( 52)            FOR TD3:=1 TO 10 DO BEGIN
( 52)               IF S.NAME[TD1,TD3]<>' ' THEN WRITE(OUTPUT,S.NAME[TD1,TD3]);
( 52)               END
( 52)            END
( 52)         ELSE WRITE(OUTPUT,M:1);
( 52)         END;(* WRITEVAL *)
( 53)  *DECK TOKEN
( 53)  PROCEDURE TOKEN
( 53)       (VAR CURS          : INTEGER;
( 53)         VAR CTYPE         : INTEGER;          (* TOKEN TYPE, 0=DELIMTP, 1=DESCTP, 2=DUMMYTP, 3=DIGITTP *)
( 53)         VAR SROW          : INTEGER;          (* SYMBOL TABLE ROW OF SYMBOL OR VALUE OF NUMBER OR ORD(DELIM) *)
( 53)         VAR ERR           : INTEGER;          (* <>0 IF ERROR *)
( 53)         VAR BUF           : CARRAY);          (* BUFFER TO SCAN *)
( 53)  LABEL
( 53)         1,
( 53)         2;
( 53)  CONST
( 53)         DELIMTP          = 0;
( 53)         DESCTP           = 1;
( 53)         DUMMYTP          = 2;
( 53)         DIGITTP          = 3;
( 53)  TYPE
( 53)         ALPHA            = SET OF 'A'..'Z';
( 53)         DIGIT            = SET OF '0'..'9';
( 53)  VAR
( 53)         TRACE,I,L,J,FCURS,LCURS: INTEGER;(*1
( 53)                    I N D U C E - 1  --  TOKEN  --  FINDROW
( 53)   *)
( 53)   (*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
( 53)                                FINDROW(B,E:INTEGER;
( 53)                    FIND A ROW IN THE SYMBOL TABLE WITH NAME IN BUF[B]..BUF[E]
( 53)   @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*)
( 53)  *CALL FINDROW
( 53)   (*1
( 53)                    I N D U C E - 1  --  TOKEN  --  FIXSYM
( 53)   *)
( 53)   (*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
( 53)                                FIXSYM(I,J:INTEGER);
( 53)                    ADD A NEW ROW TO SYMBOL TABLE
( 53)   @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*)
( 53)  *CALL FIXSYM
( 53)      (* FIXSYM *)
( 53)   (*1
( 53)                    I N D U C E - 1  --  TOKEN(MAIN CODE)
( 53)   *)
( 53)
( 53)  BEGIN  (*TOKEN*)
```

```
( 53)   42
( 53)   43        TRACE:=2;                          (* SET INTERNAL TRACE.  >2 CAUSES MESSAGES *)
( 53)   44        IF TRACE>2 THEN WRITELN(OUTPUT,'ENTERING TOKEN',CURS,CTYPE,SROW,ERR);
( 53)   45  1:    IF BUF[CURS] = '?' THEN            (* "?" IS USED TO SIGNAL END-OF-BUFFER *)
( 53)   46        BEGIN                              (* IF END OF BUFFER, *)
( 53)   47           ILINE;                          (* GET A LINE FROM THE TERMINAL *)
( 53)   48           CURS:=1;                        (* POINT CURSOR AT FIRST CHAR *)
( 53)   49           FOR I:=1 TO 100 DO              (* CLEAN THE BUFFER *)
( 53)   50           BUF[I]:=' ';       I:=1;
( 53)   51           WHILE NOT EOS(I) DO             (* SCAN THE LINE AND COPY TO BUF *)
( 53)   52           BEGIN
( 53)   53              GETCHRR(BUF[I]);       I:=I+1;
( 53)   54              END;(* WHILE *)
( 53)   55
( 53)   56           END;(*IF BUF = '?' *)
( 53)   57
( 53)   58        WHILE(BUF[CURS]=' ') AND (BUF[CURS]<>'?') DO(* SKIP LEADING BLANKS *)
( 53)   59        CURS:=CURS+1;
( 53)   60        IF BUF[CURS]='?' THEN              (* IF END-OF-LINE, GET ANOTHER *)
( 53)   61        GOTO 1;
( 53)   62        CTYPE := DELIMTP;                  (* CTYPE IS TOKEN TYPE, INIT TO DELIMITER *)
( 53)   63        FCURS := CURS;                     (* FCURS IS "FRONT" CURSOR, START OF TOKEN *)
( 53)   64  2:    IF (BUF[CURS]<='Z')AND(BUF[CURS]>='A') THEN(* IF ALPHABETIC...*)
( 53)   65        BEGIN
( 53)   66           CTYPE:=DESCTP;                  (* CHANGE TYPE TO DESCRIPTOR *)
( 53)   67           LCURS:=CURS;                    (* LCURS IS "LAST CURSOR" END OF ALPHA SYMBOL *)
( 53)   68           CURS:=CURS+1;                   (* ADVANCE CURSOR *)
( 53)   69           GOTO 2;
( 53)   70           END;
( 53)   71        IF (BUF[CURS]>='0')AND(BUF[CURS]<='9') THEN(* IF NUMERIC ...*)
( 53)   72        BEGIN
( 53)   73           IF NOT (BUF[FCURS] IN ['A'..'Z']) THEN (* AND THIS SYMBOL DIDN'T BEGIN IN ALPHABETIC *)
( 53)   74           CTYPE := DIGITTP                (* THEN IT MUST BE A PURE NUMBER *)
( 53)   75           ELSE CTYPE := DUMMYTP;          (* OTHERWISE, ITS A DUMMY VARIABLE W/ SUBSCRIPT *)
( 53)   76           CURS:=CURS+1;                   (* ADVANCE CURSOR *)
( 53)   77           GOTO 2;
( 53)   78           END;
( 53)   79        ERR:=0;                            (* NO ERRORS YET...*)
( 53)   80        CASE CTYPE OF
( 53)   81  DELIMTP: BEGIN                           (* IF ALL WE HIT WAS A DELIMITER *)
( 53)   82              CTYPE:=ORD(BUF[CURS]);       (* SET CTYPE TO INTEGER VALUE OF THE CHAR *)
( 53)   83              CURS:=CURS+1;                (* ADVANCE CURSOR *)
( 53)   84              END;
( 53)   85  DUMMYTP: BEGIN                           (* IF WE HAVE A DUMMY VARIABLE *)
( 53)   86              FINDROW(FCURS,CURS-1,I);     (* TRY TO FIND THE ENTIRE NAME IN SYMTAB *)
( 53)   87              IF I<>0 THEN                 (* IF WE FOUND IT, IT ALREADY EXISTS *)
( 53)   88              SROW:=I                      (* SET SROW TO ITS ROW *)
( 53)   89              ELSE BEGIN(* FIND ASSOC FN IN SYMTAB *)
( 53)   90
( 53)   91                 FINDROW(FCURS,LCURS,I); (* LOOK FOR "FAMILY NAME" IN THE SYMTAB *)
( 53)   92                 IF I<>0 THEN             (* IF WE FOUND IT, JUST ADD NEW DUMMY *)
( 53)   93                 BEGIN
( 53)   94                    S.NELT:=S.NELT+1;     (* ADD ONE ENTRY TO TABLE *)
( 53)   95                    SROW:=S.NELT;         (* GRAB THIS NUMBER TO RETURN AS SROW *)
( 53)   96                    FOR J:=1 TO 10 DO S.NAME[S.NELT,J]:=' ';(* CLEAN THE NAME TO BLANKS *)
( 53)   97                    FOR J:=FCURS TO CURS-1 DO(* COPY THE WHOLE NAME TO SYMTAB *)
( 53)   98                    S.NAME[S.NELT,J-FCURS+1]:=BUF[J];      S.DPNO[SROW]:=I;(* SET DPNO TO POINT TO THE "FAMILY NAME" *)
( 53)   99                    END(*I<>0*)
```

```
( 53)  100
( 53)  101                ELSE BEGIN                (* ADD BOTH A NEW FAMILY NAME AND A NEW DUMMY *)
( 53)  102                   FIXSYM(FCURS,LCURS);  (* FIXSYM WILL ADD THE FAMILY NAME *)
( 53)  103                   GOTO 1;               (* THEN WE GO BACK, RESCAN AND HANDLE IT AS ABOVE *)
( 53)  104                   END;
( 53)  105                END;(* IF I<> 0 ELSE *)
( 53)  106
( 53)  107             END;(*CASE DUMYTP *)
( 53)  108
( 53)  109  DESCTP:  BEGIN
( 53)  110             FINDROW(FCURS,CURS-1,I);    (* ITS AN ORDINARY DESCRIPTOR, FIND IT IN SYMTAB *)
( 53)  111             IF I=0 THEN                 (* IF NOT FOUND, *)
( 53)  112             BEGIN
( 53)  113                FIXSYM(FCURS,CURS-1);    (* ADD IT *)
( 53)  114                GOTO 1;
( 53)  115                END
( 53)  116             ELSE SROW:=I;               (* IF FOUND, RETURN SROW *)
( 53)  117             END;(*CASE DESCTP *)
( 53)  118
( 53)  119  DIGITTP: BEGIN                         (* ITS A NUMERIC FIELD *)
( 53)  120             SROW:=0;                    (* RETURN VALUE OF NUMBER AS SROW *)
( 53)  121             FOR I:=FCURS TO CURS-1 DO   (* CONVERT TO BINARY *)
( 53)  122             SROW:=SROW*10+ ORD(BUF[I])-ORD('0');
( 53)  123             IF SROW>MNVAL THEN BEGIN    (* IF THE VALUE IS BIGGER THAN MAX REFERENCE VALUE *)
( 53)  124                WRITELN(OUTPUT,'THE VALUE ',SROW,' IS BEYOND THE MAXIMUM ', 'DOMAIN LIMITS (MNVAL=',MNVAL:2,').');
( 53)  125                ERR:=1;
( 53)  126                END;                     (* THE ONLY NUMBERS WE SCAN ARE REFERENCE VALUES SO THIS WORKS *)
( 53)  127             END
( 53)  128          END;(* CASE STMT *)
( 53)  129
( 53)  130       CASE ERR OF                       (* WERE THERE ANY ERRORS *)
( 53)  131  1:       WRITELN(OUTPUT,'INVALID CHARACTER OR VALUE');
( 53)  132  0:
( 53)  133          END;(*CASE STMT*)
( 53)  134
( 53)  135       IF TRACE>2 THEN WRITELN(OUTPUT,'LEAVING TOKEN',CURS,CTYPE,SROW,ERR);
( 53)  136       END;  (* TOKEN *)
( 54)    1  *DECK FINDROW
( 54)    2  PROCEDURE FINDROW
( 54)    3       (B,E             : INTEGER;
( 54)    4        VAR TMP         : INTEGER);
( 54)    5  LABEL
( 54)    6       1,
( 54)    7       2;
( 54)    8  VAR
( 54)    9       J,I              : INTEGER;(* FIND ROW IN SYMTAB WHICH MATCHES BUF, PUT IN I*
( 54)   10      *)
( 54)   11
( 54)   12  BEGIN
( 54)   13       IF TRACE>2 THEN                   (* PRINT TRACE INFORMATION FOR DEBUGGING *)
( 54)   14       WRITELN(OUTPUT,'ENTERING FINDROW',B,E);
( 54)   15       IF E-B+1>10 THEN E:=B+9;       (*  TRUNCATE SYMBOL IF TOO LONG *)
( 54)   16       FOR I:=1 TO S.NELT DO BEGIN
( 54)   17          FOR J:=1 TO E-B+1 DO           (* SEARCH FOR THIS SYMBOL IN THE SYMTAB *)
( 54)   18          IF S.NAME[I,J]<>BUF[B-1+J] THEN GOTO 1;
( 54)   19          IF E-B+2<=10 THEN              (* IF IT MATCHES WITH LESS THAN 10 CHARS *)
( 54)   20             IF S.NAME[I,E-B+2]<>' ' THEN GOTO 1; (*MAKE SURE THEYRE SAME LENGTH*)
( 54)   21          GOTO 2;
```

```
( 54)   22  1:        ;
( 54)   23            END;(* FOR I *)
( 54)   24
( 54)   25        I:=0;
( 54)   26  2:    TMP:=I;                          (* RETURN 0 IF NO MATCH, ELSE PNO ROW *)
( 54)   27        END; (* FINDROW *)
( 55)    1  *DECK FIXSYM
( 55)    2  PROCEDURE FIXSYM
( 55)    3       (I,J              : INTEGER);
( 55)    4  VAR
( 55)    5        K,L              : INTEGER;
( 55)    6        C                : CHAR;
( 55)    7  BEGIN
( 55)    8        IF TRACE>2 THEN                  (* PRINT TRACE INFORMATION DURING DEBUGGING *)
( 55)    9        WRITELN(OUTPUT,'ENTERING FIXSYM',I,J);
( 55)   10        IF J-I+1>10 THEN BEGIN(* SYMBOL IS TOO LONG, TRUNCATE IT *)
( 55)   11           WRITELN(OUTPUT,'SYMBOL IS TOO LONG, TRUNCATED TO 10 CHARS');      J:=I+9;
( 55)   12           END;(* ADD ROW TO STAB OR ELSE REPLACE DESC IN BUF *)
( 55)   13
( 55)   14        S.NELT:=S.NELT+1;                (* ADD 1 TO ROW COUNT *)
( 55)   15        IF S.NELT>SYMSZE THEN            (* IS THERE ROOM ? *)
( 55)   16        WRITE(OUTPUT,'SYMBOL TABLE OVERFLOW, ');
( 55)   17        FOR K:=I TO J DO                 (* COPY THE SYMBOL INTO THE TABLE *)
( 55)   18        S.NAME[S.NELT,K-I+1]:=BUF[K];      S.PNO[S.NELT]:=S.NELT;(* SET THE PNO ENTRY TO ITSELF *)
( 55)   19        CURS:=I;                         (* REPOSITION CURSOR TO BEGINNING OF SYMBOL *)
( 55)   20        IF TRACE>2 THEN WRITELN(OUTPUT,'LEAVING FIXSYM');
( 55)   21        END;
( 56)    1  *DECK VLINT
( 56)    2  PROCEDURE VLINT;        (* ARGS ARE G:GPTR; VAR ERR:INTEGER;VAR ES:INTEGER;PRODNUM:INTEGER;A:ARITHPTR *)
( 56)    3                                         (* PRODNUM IS # OF PRODUCTION TO START PARSE *)
( 56)    4                                         (* A IS ARITHSTACK IF PARSING ARITH EXPRESSION *)
( 56)    5  LABEL
( 56)    6        11,
( 56)    7        10,
( 56)    8        1,
( 56)    9        2,
( 56)   10        3,
( 56)   11        4,
( 56)   12        5,
( 56)   13        98,
( 56)   14        99,
( 56)   15        12;
( 56)   16  VAR
( 56)   17        VTOP             : INTEGER;      (* TOP OF VSTK *)
( 56)   18        FTOP             : INTEGER;      (* TOP OF FSTK *)
( 56)   19        STOP             : INTEGER;      (* TOP OF SSTK *)
( 56)   20        PTOP             : INTEGER;      (* TOP OF PSTK *)
( 56)   21        OTOP             : INTEGER;      (* TOP OF OPERATION STACK OPSTK *)
( 56)   22        PROD             : INTEGER;      (* CURRENT PRODUCTION (NEGATIVE) *)
( 56)   23        LOC              : INTEGER;      (* CURRENT ROW OF CURRENT RHS *)
( 56)   24        CURS             : INTEGER;      (* CURRENT CURSOR INTO BUF *)
( 56)   25        CTYPE            : INTEGER;      (* TOKEN TYPE:  0=DELIMINTER 1=DESCRIPTOR 2=DUMMY 3=NUMBER *)
( 56)   26        SROW             : INTEGER;      (* ROW OF SYMTAB, RETURNED BY TOKEN *)
( 56)   27        GDESC            : INTEGER;      (* ?? *)
( 56)   28        GTOP             : INTEGER;      (* CURRENT ROW IN GSTRUC BEING BUILT *)
( 56)   29        I,J,K,L          : INTEGER;      (* MISC INTEGERS *)
( 56)   30        ANO              : INTEGER;      (* NUMBER OF ARGUMENTS FOR CURRENT DESCRIPTOR *)
( 56)   31        TRACE            : INTEGER;      (* CONTROLS INTERNAL TRACING.  TRACE IF > 2 *)
```

```
( 56)   32        MAXLOC,MAXPROD : INTEGER;          (* PRODUCTION WHICH WE MADE THE MOST
( 56)   33                                              TOWARD COMPLETING--USED FOR ERROR MSG *)
( 56)   34        VSTK           : ARRAY[1..GSIZE] OF VALTP;(* VALUE STACK, USED TO COMPUTE VALUES OF REFERENCE *)
( 56)   35        SSTK           : ARRAY[1..150] OF INTEGER;(* SEMANTIC STACK -- HAS SCANNED TERMINALS AND PRODS *)
( 56)   36        FSTK           : ARRAY[1..150] OF INTEGER;(* DESCRIPTORS AND DUMMY VARIABLES *)
( 56)   37        PSTK           : ARRAY[1..200] OF INTEGER;(* NON-TERMINALS ALREADY TRIED *)
( 56)   38        OPSTK          : ARRAY[1..MAXASTACK] OF ACTIONCODE;(* OPERATION CODE STACK *)
( 56)   39        BUF            : CARRAY;(* BUFFER FOR CURRENT LINE *)
( 56)   40        DONE           : BOOLEAN;          (* SEMANTIC ROUTINES THINK THEY'RE DONE IF TRUE *)
( 56)   41    (*1
( 56)   42                  I N D U C E - 1  --  VLINT  --  PROCESS
( 56)   43    *)
( 56)   44    *CALL PROCESS
( 56)   45        (*PROCESS*)
( 56)   46    (*1
( 56)   47                  I N D U C E - 1  --  VLINT (MAIN CODE)
( 56)   48    *)
( 56)   49
( 56)   50    BEGIN
( 56)   51        IF PRODNUM=VLRULE THEN BEGIN      (* IF PARSING VL RULE *)
( 56)   52           IF INFILE = 0 THEN               (* IF READING FROM TTY *)
( 56)   53           WRITELN(OUTPUT,'RULE  #',G^.RNO:5); (* PROMPT PRETTY *)
( 56)   54           FOR I:=1 TO GSIZE DO G^.PNO[I]:=0;(* CLEAN THE GRAPH *)
( 56)   55           FOR I:=1 TO GSIZE DO             (* CLEAN THE LINKS *)
( 56)   56           FOR J:=1 TO MLNK DO G^.LNK[I,J]:=0;
( 56)   57           END;(*IF VLRULE *)
( 56)   58        CURS:=101;                          (* FORCE END OF BUFFER,  CURSOR=101 POINTS TO ? *)
( 56)   59        BUF[101]:='?';                        TRACE:=2;(* SET INTERNAL TRACE.  IF > 2 CAUSES TRACE OF PARSE *)
( 56)   60        FOR I:=1 TO 150 DO                 (* CLEAN THE SEMANTIC STACK *)
( 56)   61        SSTK[1]:=0;       VTOP:=0;         (* INITIALIZE THE STACK POINTERS *)
( 56)   62        FTOP:=0;          GTOP:=0;
( 56)   63        OTOP:=0;          STOP:=1;         (* SSTK CAN HANDLE A ZERO TOP ELEMENT --NOT A TRUE STACK *)
( 56)   64        PTOP:=0;          MAXLOC := 0;
( 56)   65        MAXPROD := PRODNUM;        (*  INITIALIZE OUR "BEST PRODUCTION POINTERS"
( 56)   66                                TO POINT TO "VL DECISION RULE" *)
( 56)   67    11: PROD:=-PRODNUM;                       (* POINT TO FIRST PRODUCTION RULE (PROD IS ALWAYS NEG ) *)
( 56)   68        LOC:=1;                            (* POINT TO FIRST RHS ENTRY *)
( 56)   69        WITH PTBL DO BEGIN
( 56)   70    1:     IF(RHS[-PROD,LOC]<>0) AND (LOC>MAXLOC) THEN (* IS THIS A NEW MAXIMUM POSITION? *)
( 56)   71           BEGIN
( 56)   72              MAXLOC := LOC;       MAXPROD := -PROD;(* THIS IS THE BEST PROD SO FAR *)
( 56)   73              END;
( 56)   74           IF SSTK[STOP]=0 THEN               (* IS THERE NOTHING ON THE SSTK ? *)
( 56)   75           BEGIN
( 56)   76              TOKEN(CURS,CTYPE,SROW,ERR,BUF);  (* YES, GET A NEW TOKEN *)
( 56)   77              SSTK[STOP]:=CTYPE;             (* PUSH THE CTYPE ON SSTK *)
( 56)   78              IF ERR <> 0 THEN               (* IF THERE'S AN ERROR, GET OUT, REEXECUTE *)
( 56)   79              GOTO 99;
( 56)   80              END;
( 56)   81           IF (RHS[-PROD,LOC]<0) AND (RHS[-PROD,LOC]<>SSTK[STOP])
( 56)   82              THEN BEGIN(* IF ITS A NON-TERMINAL, BUT NOT THE RIGHT ONE *)
( 56)   83                     (* PUSH PROD AND LOC *)
( 56)   84
( 56)   85              PSTK[PTOP+1]:=PROD;       PSTK[PTOP+2]:=LOC;
( 56)   86              IF TRACE>2 THEN WRITELN(OUTPUT,'PUSH',PROD,LOC);
( 56)   87              PTOP:=PTOP+2;       PROD:=RHS[-PROD,LOC];(* GO DOWN THE TREE TO NEXT PRODUCTION *)
( 56)   88              LOC:=1;                        (* START WITH FIRST RHS ENTRY *)
( 56)   89              GOTO 1;
```

```
( 56)   90              END;(* IF --- AND --- THEN*)
( 56)   91
( 56)   92           IF RHS[-PROD,LOC]<>0 THEN     (* IF WE AREN'T DONE WITH THIS PRODUCTION, *)
( 56)   93              IF RHS[-PROD,LOC]=SSTK[STOP] THEN(* AND WE HAVE A MATCH *)
( 56)   94              BEGIN     (* ENTRY IN PT MATCHES TOKEN *)
( 56)   95
( 56)   96                 STOP:=STOP+1;           (* PUSH THE TOKEN *)
( 56)   97                 LOC:=LOC+1;             (* AND PARSE THE NEXT RHS ENTRY *)
( 56)   98                 GOTO 1;
( 56)   99                 END(* RHS = SSTK *)
( 56)  100
( 56)  101              ELSE BEGIN(* IT IS A TERMINAL AND IT DIDN'T MATCH OR ELSE WE ARE AT THE END OF A PRODUCTION *)
( 56)  102                  (* ENTRY DOES NOT MATCH SSTK*)
( 56)  103
( 56)  104                 STOP:=STOP-(LOC-1);     (* BACK UP THE BEGINNING OF THIS PRODUCTION *)
( 56)  105                 PROD:=PROD-1;           (* AND TRY THE NEXT ROW OF THE PT *)
( 56)  106                 LOC:=1;                 (* FIRST RHS ELEMENT *)
( 56)  107                 IF TRACE>2 THEN WRITELN(OUTPUT,'NOMATCH',PROD+1);
( 56)  108                 IF ABS(PROD)>MAXPTSIZE THEN GOTO 12;(* IF THERE IS NO NEXT ROW...*)
( 56)  109  10:            IF CONT[-PROD] THEN GOTO 1;(* ELSE, IF THIS ROW IS A CONTINUATION OF LAST *)
( 56)  110  12:            BEGIN                   (* THERE IS NO NEXT ROW, WE MUST GIVE UP ON THIS NON-TERMINAL *)
( 56)  111                    PTOP:=PTOP-2;       (* BACK UP TO PREVIOUS CLASS OF ALTERNATIVES *)
( 56)  112                    IF PTOP=-2 THEN     (* ARE THERE ANY MORE? *)
( 56)  113                    GOTO 98;            (* NO, BAD SYNTAX *)
( 56)  114                    STOP:=STOP-(PSTK[PTOP+2]-1);(* YES, BACK UP STK TO START OF CLASS *)
( 56)  115                    PROD:=PSTK[PTOP+1]-1;(* AND BACK UP TO THE LAST PRODUCTION + 1 *)
( 56)  116                    GOTO 10;
( 56)  117                    END;
( 56)  118                 END;(* IF RHS = SSTK *)
( 56)  119
( 56)  120           BEGIN  (* EXECUTE PROC *)
( 56)  121
( 56)  122              PROCESS(DONE);             (* WE HAVE A FINISHED PRODUCTION, DO IT *)
( 56)  123              IF -PROD=MAXPROD THEN BEGIN(* IF WE FINISHED OUR "BEST" PRODUCTION, ERASE IT *)
( 56)  124                 MAXPROD:=PRODNUM;       (* AND FIND ANOTHER NEXT TIME AT 1: *)
( 56)  125                 MAXLOC:=0;
( 56)  126                 END;(*  IF  *)
( 56)  127              IF DONE THEN               (* IF SEMANTIC RULES THINK WE'RE DONE *)
( 56)  128              GOTO 2;                    (* QUIT *)
( 56)  129              IF TRACE>2 THEN WRITELN(OUTPUT,'PROC',PROD);(*REPLACE LOC-1 ENTRIES IN SSTK WITH PROD *)
( 56)  130
( 56)  131              STOP:=STOP-(LOC-1);        (* REMOVE THE PARSED TERMINALS WITH A NON-TERMINAL *)
( 56)  132              FOR J:=STOP+1 TO 150 DO    (* RIPPLE DOWN THE REMAINING ELEMENTS *)
( 56)  133              IF J+LOC-2<=150 THEN SSTK[J]:=SSTK[J+LOC-2];
( 56)  134              WHILE CONT[-PROD] DO       (* SKIP REMAINING ALTERNATIVES FOR THIS CLASS *)
( 56)  135              PROD:=PROD+1;       SSTK[STOP]:=PROD;(* AND PUSH THE LAST ONE ONTO SSTK *)
( 56)  136              PTOP:=PTOP-2;
( 56)  137              IF PTOP=-2 THEN            (* WE THINK WE'RE DONE, BUT SEMANTIC RULES THINK WE AREN'T *)
( 56)  138              GOTO 2;
( 56)  139              PROD:=PSTK[PTOP+1];        (* GO TRY THE NEXT PROD *)
( 56)  140              LOC:=PSTK[PTOP+2]+1;       (* AT THE NEXT LOC *)
( 56)  141              IF TRACE>2 THEN WRITELN(OUTPUT,'POP',PROD,LOC,STOP);
( 56)  142              STOP:=STOP+1;
( 56)  143              GOTO 1;
( 56)  144              END;
( 56)  145           END;(*WITH*)
( 56)  146
( 56)  147        GOTO 99;
```

```
( 56)  148  98:   (*  PRINT OUT INVALID SYNTAX MESSAGES  *)
( 56)  149        WRITE(OUTPUT,'INCORRECT SYNTAX -- INSTEAD OF ');
( 56)  150        IF MAXPROD=1 THEN WRITE(OUTPUT,'A VL DECISION RULE')
( 56)  151        ELSE IF PTBL.RHS[MAXPROD,MAXLOC]>0 THEN WRITE(OUTPUT,'"',CHR(PTBL.RHS[MAXPROD,MAXLOC]),'"')(* DELIMITER CHAR *)
( 56)  152        ELSE BEGIN
( 56)  153           J:=1;
( 56)  154           WHILE (ORD(PTBL.NAME[ABS(PTBL.RHS[MAXPROD,MAXLOC]),J])<>0) AND (J<=60) DO BEGIN
( 56)  155              WRITE(OUTPUT,PTBL.NAME[ABS(PTBL.RHS[MAXPROD,MAXLOC]),J]);(* PRODUCTION *)
( 56)  156              J:=J+1;
( 56)  157              END;(* WHILE *)
( 56)  158           END; (*  ELSE  *)
( 56)  159        WRITELN(OUTPUT);       WRITE(OUTPUT,'YOU TYPED ');
( 56)  160        IF CTYPE IN [1..3] THEN CASE CTYPE OF
( 56)  161  1:       BEGIN
( 56)  162              WRITE(OUTPUT,'FUNCTION SYMBOL ');
( 56)  163              FOR J:=1 TO 10 DO IF S.NAME[SROW,J]<>' ' THEN WRITE(OUTPUT,S.NAME[SROW,J]);
( 56)  164              END;
( 56)  165  2:       BEGIN
( 56)  166              WRITE(OUTPUT,'VARIABLE ');
( 56)  167              FOR J:=1 TO 10 DO IF S.NAME[SROW,J]<>' ' THEN WRITE(OUTPUT,S.NAME[SROW,J]);
( 56)  168              END;
( 56)  169  3:       WRITE(OUTPUT,'A NUMBER (',SROW:4,')');
( 56)  170           END    (* CASE CTYPE *)
( 56)  171        ELSE WRITE(OUTPUT,'"',CHR(CTYPE),'"');     (* PRINT THE OFFENDING CHAR *)
( 56)  172        WRITELN(OUTPUT,'.');                       (* FINISH MY SENTENCE *)
( 56)  173        ERR:=1;
( 56)  174        IF CTYPE <= 2 THEN                (* IF CTYPE IS NOT A NUMBER OR DELIM *)
( 56)  175        GOTO 99;                          (* ABORT THIS AND REEXECUTE *)
( 56)  176        ERR:=0;                           (* RESET ERROR AND TRY TO CONTINUE *)
( 56)  177        STOP:=1;                          (* RESET SSTK *)
( 56)  178        WHILE SSTK[STOP+1]<>0 DO          (* SCAN THE CURRENT SSTK *)
( 56)  179        BEGIN
( 56)  180           IF SSTK[STOP]<0 THEN           (* IF ITS A NON-TERMINAL *)
( 56)  181           WHILE PTBL.CONT[-SSTK[STOP]] DO   (* SKIP ITS CONTINUATIONS *)
( 56)  182           SSTK[STOP]:=SSTK[STOP]+1;      STOP:=STOP+1;
( 56)  183           END;
( 56)  184        SSTK[STOP]:=0;                    (* CLEAR OUT THE LAST (MOST RECENT) TOKEN *)
( 56)  185        FOR J:=1 TO CURS-1 DO WRITE(OUTPUT,BUF[J]);(* PRINT THE BUF AS IT STANDS *)
( 56)  186        WRITELN(OUTPUT);       WRITELN(OUTPUT,'RETYPE LAST CHARACTER');       INFILE:=0;(* SET INPUT TO TELETYPE *)
( 56)  187        ILINE;                            (* AND GIVE THE GUY ANOTHER SHOT *)
( 56)  188        READ(IFILE,BUF[CURS-1]);                                              CURS:=CURS-1;
( 56)  189        I:=1;
( 56)  190        WHILE NOT EOLN(IFILE) DO BEGIN    (* ?? RIPPLE ONE CHAR THRU BUFFER??? READ BUFFER??*)
( 56)  191           FOR J:=CURS+I TO 99 DO BUF[J+1]:=BUF[J];
( 56)  192           READ(IFILE,BUF[CURS+I]);       (* GET NEXT BUFFER ELEMENT *)
( 56)  193           I:=I+1;
( 56)  194           END;
( 56)  195        PTOP:=0;       STOP:=1;           (* FORCE A QUICK REPARSE OF ENTIRE TREE... *)
( 56)  196        GOTO 11;
( 56)  197  2:    IF (PSTK[1] < -3) AND (PRODNUM=VLRULE) THEN   (* SEMANTIC AND SYNTACTIC ROUTINES DISAGREE...*)
( 56)  198        GOTO 98;                          (* BAD SYNTAX *)
( 56)  199      (* IF RESTRICTION, THEN PLACE CONS AT END OF G
( 56)  200     AND DELETE INCOMING LINKS*)
( 56)  201        IF CHRR='R' THEN BEGIN
( 56)  202           I:=1;
( 56)  203           WHILE G^.LNK[GTOP,I]<>0 DO BEGIN
( 56)  204              G^.LNK[GSIZE,I]:=G^.LNK[GTOP,I];      J:=1;
( 56)  205              WHILE G^.LNK[G^.LNK[GTOP,I],J]<>0 DO J:=J+1;
```

```
( 56)  206              G^.LNK[G^.LNK[GTOP,I],J-1]:=0;       G^.LNK[GTOP,I]:=0;       I:=I+1;
( 56)  207              END;(*WHILE G^...<>0*)
( 56)  208
( 56)  209            G^.VBL[GSIZE]:=G^.VBL[GTOP];  (* COPY THE INFORMATION TO THE "LAST" ENTRY IN G *)
( 56)  210            G^.ORDIRR[GSIZE]:=G^.ORDIRR[GTOP];     G^.VAL[GSIZE]:=G^.VAL[GTOP];
( 56)  211            G^.PNO[GSIZE]:=G^.PNO[GTOP];
( 56)  212            END;(*IF CHRR='R'*)
( 56)  213
( 56)  214  99:    ;
( 56)  215        END;(* VLINT *)
( 57)    1  *DECK PROCESS
( 57)    2  PROCEDURE PROCESS
( 57)    3       (VAR DONE       : BOOLEAN);        (* PERFORM SEMANTIC PROCESSING *)
( 57)    4  LABEL
( 57)    5        1,
( 57)    6        2,
( 57)    7        3,
( 57)    8        4,
( 57)    9        5;
( 57)   10  VAR
( 57)   11        I,J             : INTEGER;(*
( 57)   12                  I N D U C E - 1  --  VLINT  --  PROCESS  --  TESTGTOP
( 57)   13   *)
( 57)   14  *CALL TESTGTOP
( 57)   15   (*1
( 57)   16                  I N D U C E - 1  --  VLINT  --  PROCESS  --  DUMPROC
( 57)   17
( 57)   18   *)
( 57)   19  *CALL DUMPROC
( 57)   20      (*DUMPROC*)
( 57)   21   (*1
( 57)   22                  I N D U C E - 1  --  VLINT --  PROCESS  --  PARSEARITH
( 57)   23   *)
( 57)   24  *CALL PARSEARITH
( 57)   25   (*1
( 57)   26                  I N D U C E - 1  --  VLINT --  PROCESS(MAIN CODE)
( 57)   27   *)
( 57)   28
( 57)   29  BEGIN
( 57)   30        DONE:=FALSE;                        (* NOT DONE YET *)
( 57)   31        CASE PTBL.SRULE[-PROD] OF           (* CASE THE SEMANTIC RULE FOR THIS PRODUCTION *)
( 57)   32            (* DESC *)
( 57)   33
( 57)   34            (* SROW HAS LOC IN STAB OF DESC,ALOC NODE FOR DESC
( 57)   35             *)
( 57)   36
( 57)   37  21,22,23,24,25,26,27,28,29,30,31,32:
( 57)   38            PARSEARITH;                                 (* HANDLE ARITH PARSING *)
( 57)   39
( 57)   40  16,17,18,19,33:
( 57)   41            DUMPROC;                      (* GO TO THE ABOVE PROCEDURE, KEEP THIS PROCEDURE SMALL*)
( 57)   42  15:       BEGIN(* FIND DUMMY IN GRAPH, PUSH LOC IN GRAPH *)
( 57)   43
( 57)   44               IF CHRR<>'E' THEN           (* IF WE'RE NOT DOING STRUCTURE *)
( 57)   45               FOR I:=1 TO GTOP DO         (* SEARCH THE GRAPH *)
( 57)   46                  IF G^.DUMNUM[I]=SROW THEN(* FOR THIS DUMMY VARIABLE *)
( 57)   47                  GOTO 3;
( 57)   48               GTOP:=GTOP+1;               (* IF NOT F   D, ADD IT *)
```

```
( 57)   49              TESTGTOP;                    (* MAKE SURE WE DONT EXCEED GRAPH*)
( 57)   50              I:=GTOP;                          G^.DUMNUM[I]:=SROW;(* POINT DUMNUM TO SYMTAB ROW OF FULL NAME *)
( 57)   51              G^.PNO[I]:=S.DPNO[SROW];     (* POINT PNO TO "FAMILY" NAME OF SYMBOL TABLE *)
( 57)   52              G^.DPNO[I]:=SROW;       (* POINT DPNO IN G TO THE TRUE ROW SO THAT
( 57)   53                                         COST FUNCTION(5) CAN ACCESS IT *)
( 57)   54              G^.VBL[I]:=TRUE;             (* THIS IS A VARIABLE *)
( 57)   55              G^.ORDIRR[GTOP]:=TRUE;       (* ORDER IS IRRELEVANT *)
( 57)   56              G^.VAL[I]:=[0..MNVAL];       (* INITIALLY HAS * ALL VALUES *)
( 57)   57  3:          FTOP:=FTOP+1;                (* PUSH THE GRAPH ROW ONTO FSTK *)
( 57)   58              FSTK[FTOP]:=I;       ANO:=ANO+1;(* ONE MORE ARGUMENT FOR CURRENT FUNCTION *)
( 57)   59              IF CHRR='E' THEN             (* IF WE ARE ENTERING STRUCTURE, FORCE *)
( 57)   60              S.VTYPE[G^.PNO[FSTK[I]]]:=3;(* STRUCTURED DOMAIN FOR CURRENT DESCRIPTOR *)
( 57)   61              END;(* AREST *)
( 57)   62
( 57)   63             (* POP VALUE AND DUMMY VAR, FIND DUMMY VAR IN G,SE
( 57)   64              T ARG*)
( 57)   65
( 57)   66  14,13:    BEGIN                          (* A SELECTOR INVOLVING ONLY A DUMMY VAR *)
( 57)   67              G^.VAL[FSTK[FTOP]]:=VSTK[VTOP];(* THE VALUE IS THE CURRENT VALUE *)
( 57)   68              VTOP:=VTOP-1;                (* POP VALUE AND FSTK POINTER *)
( 57)   69              FTOP:=FTOP-1;
( 57)   70              END;(* ALIST *)
( 57)   71
( 57)   72             (* LINK DUMMY ON STK TO G DESC, J IS DUMMY DESC LO
( 57)   73              C*)
( 57)   74
( 57)   75  20,12,11:
( 57)   76            BEGIN                          (* A DEPENDENT VARIABLE LIST HAS BEEN HIT *)
( 57)   77              J:=FSTK[FTOP];               (* REMEMBER TOP OF FSTK *)
( 57)   78              IF PTBL.SRULE[-PROD]=20 THEN(* IF IT HAS ORDIRR "," BETWEEN VARS *)
( 57)   79              BEGIN  (* G^.PNO[FSTK[1]]:= -ABS(G^.PNO[FSTK[1]]);*)
( 57)   80
( 57)   81                 G^.ORDIRR[FSTK[1]]:=TRUE;(* SET ORDIRR FOR THIS DESCRIPTOR *)
( 57)   82                 END;
( 57)   83              G^.LNK[FSTK[1],ANO]:=J;      (* LINK THIS VAR TO THE DESCRIPTOR *)
( 57)   84              IF PTBL.SRULE[-PROD]<>20 THEN(* IF ITS NOT ORDIRR *)
( 57)   85                 IF S.NARG[G^.PNO[FSTK[I]]]<ANO THEN(* AND NUMBER OF ARGS IS TOO SMALL *)
( 57)   86                 S.NARG[G^.PNO[FSTK[1]]]:=ANO;(* FIX IT.  ORDIRR HAVE VARIABLE # OF ARGS *)
( 57)   87              ANO:=ANO-1;                  (* DECREMENT ANO--IT IS USED AS AN INDEX *)
( 57)   88              FTOP:=FTOP-1;                (* POP THE DUMMY OFF FSTK *)
( 57)   89              FOR I:=1 TO MLNK DO         (* SET THE FIRST NON-ZERO LINK TO POINT BACK *)
( 57)   90              IF G^.LNK[J,I]=0 THEN        (* TO THE DESCRIPTOR *)
( 57)   91              GOTO 5;
( 57)   92  5:          G^.LNK[J,I]:=FSTK[1];
( 57)   93              END;(* RNG *)
( 57)   94
( 57)   95  10:       BEGIN                          (* A SINGLE NUMBER IN THE REFERENCE *)
( 57)   96               (*ALLOCATE NEW VAL ELT, PUT DIGIT IN THIS *)
( 57)   97
( 57)   98              VTOP:=VTOP+1;                (* PUSH IT ONTO VALUE STACK *)
( 57)   99              VSTK[VTOP]:=[-FSTK[FTOP]];
( 57)  100              FTOP:=FTOP-1;                (* AND POP DUMMY OFF THE VARIABLE STACK *)
( 57)  101              END;(*RNG  *)
( 57)  102
( 57)  103  9:        BEGIN(* INTERVAL VARIABLE *)
( 57)  104
( 57)  105              S.VTYPE[G^.PNO[FSTK[1]]]:=2;(* SET THE VTYPE TO INTERVAL *)
( 57)  106              VTOP:=VTOP+1;                (* PUSH ONTO THE VALUE STACK *)
```

```
( 57)  107              VSTK[VTOP]:=[];                 (* NULL SET *)
( 57)  108              FOR I:=-FSTK[FTOP-1] TO -FSTK[FTOP] DO
( 57)  109                 VSTK[VTOP]:=VSTK[VTOP]+[I];(* ADD THE NEGATIVE OF THE INTERVENING VALUES TO VSTK *)
( 57)  110              FTOP:=FTOP-2;                   (* POP TWO ELEMENTS OFF FUNCTION STACK *)
( 57)  111              END;(*INTERVAL VARIABLE**)
( 57)  112
( 57)  113  8:        BEGIN                             (* A LIST OF NUMBERS *)
( 57)  114                (* PUT DIGIT IN THE VAL SET *)
( 57)  115
( 57)  116              VSTK[VTOP]:=VSTK[VTOP]+[-FSTK[FTOP]];(* ADD THE TOP FSTK ELEMENT TO THE VALUE LIST *)
( 57)  117              FTOP:=FTOP-1;                   (* AND POP THE FSTK *)
( 57)  118              END;(* SEL *)
( 57)  119
( 57)  120  6,7:      BEGIN                             (* NORMAL FUNCTION SELECTOR *)
( 57)  121                (* PUT VAL IN FSTK[1], PLACE IN G *)
( 57)  122
( 57)  123              G^.VAL[FSTK[1]]:=VSTK[VTOP];(* POP THE VALUE INTO THE RIGHT GRAPH ROW *)
( 57)  124              VTOP:=VTOP-1;       FTOP:=FTOP-1;(* AND POP THE FUNCTION OFF THE FSTK *)
( 57)  125              END;(* VLFORM *)
( 57)  126
( 57)  127  5,4:      BEGIN                             (* COMPLETED CONJUNCTION OF SELECTORS *)
( 57)  128                (* NOTHING *)
( 57)  129
( 57)  130              ;
( 57)  131              END;(* ERULE *)
( 57)  132
( 57)  133  3:        BEGIN                             (* COMPLETED VL DECISION RULE *)
( 57)  134                (* FIX SET OF THE GRAPH *)
( 57)  135
( 57)  136              G^.ESET:=G^.VAL[FSTK[1]];  (* GET THE RIGHT EVENT SETS *)
( 57)  137              FOR J:=1 TO 10 DO          (* DETERMINE THE HIGHEST EVENT SET *)
( 57)  138              IF J IN G^.ESET THEN K:=J; (* AND RETURN IT AS ES *)
( 57)  139              ES:=K;       DONE:=TRUE;   (* WE SHOULD BE DONE NOW--SYNTAX ROUTINES CHECK *)
( 57)  140              GOTO 2;
( 57)  141              END;(*VVERULE *)
( 57)  142
( 57)  143  2:        ;(*VVERULE*)
( 57)  144
( 57)  145  1:        BEGIN                             (*  VL DECISION RULE WITH A LEADING COEFFICIENT *)
( 57)  146                (* POP DIGIT, PUT INTO GRAPH *)
( 57)  147
( 57)  148              G^.COEF:=-FSTK[1];              (* ADD COEFFICIENT TO STACK *)
( 57)  149              END
( 57)  150            END;(* CASE STMT *)
( 57)  151
( 57)  152  2:      ;
( 57)  153        END;
( 58)    1  *DECK TESTGTOP
( 58)    2  PROCEDURE TESTGTOP;(* PURPOSE:  TO GIVE ERROR MESSAGE IF WE EXCEED GSIZE *)
( 58)    3  BEGIN
( 58)    4        IF GTOP > GSIZE THEN WRITELN(OUTPUT,'RULE ',G^.RNO:3,' IS TOO LARGE TO FIT IN PROGRAM', '-- GSIZE = ',GSIZE:3);
( 58)    5        END;(*TESTGTOP*)
( 59)    1  *DECK DUMPROC
( 59)    2  PROCEDURE DUMPROC;                         (* PERFORM SEMANTICS ON DUMMY VARIABLES ETC *)
( 59)    3  BEGIN
( 59)    4        CASE PTBL.SRULE[-PROD] OF           (* CASE THE SEMANTIC RULE FOR THIS PRODUCTION *)
( 59)    5  19:      BEGIN                             (*  "*" ENCOUNTERED, SET VALUE TO *  *)
( 59)    6              VTOP:=VTOP+1;                 (* ADD ANOTHER ELEMENT TO VSTK *)
```

```
( 59)    7              VSTK[VTOP]:=[0..MNVAL];     (* MAKE THAT ELEMENT * *)
( 59)    8              END;
( 59)    9  18:       BEGIN                         (* SIMPLE PREDICATE SELECTOR *)
( 59)   10              G^.VAL[FSTK[1]]:=[1];       (* SET VALUE TO TRUE *)
( 59)   11              FTOP:=FTOP-1;               (* POP OFF AN FSTK ELEMENT *)
( 59)   12              IF S.MVAL[ABS(G^.PNO[FSTK[1]])]>1 THEN(* FORCE THIS TO BE A PREDICATE *)
( 59)   13              S.MVAL[ABS(G^.PNO[FSTK[1]]) ]:=1;
( 59)   14              IF S.NVAL[ABS(G^.PNO[FSTK[1]])]<1 THEN(* BY SETTING MVAL=NVAL=1 *)
( 59)   15              S.NVAL[ABS(G^.PNO[FSTK[1]]) ]:=1;
( 59)   16              S.EVAL[ABS(G^.PNO[FSTK[1]])]:=S.NVAL[ABS(G^.PNO[FSTK[1]])];(* SET EVAL PROPERLY *)
( 59)   17              END;
( 59)   18  17:       BEGIN                         (* PROCESS A FUNCTION SYMBOL *)
( 59)   19              GTOP:=GTOP+1;               (* ADD A LINE TO THE GRAPH *)
( 59)   20              TESTGTOP;                   (* MAKE SURE WE DONT EXCEED GRAPH *)
( 59)   21              FSTK[1]:=GTOP;              (* SET FSTK[1] TO POINT TO CURRENT GSTK TOP *)
( 59)   22              FTOP:=FTOP+1;               (* PUCH A NEW (EMPTY) ELEMENT ONTO GSTK--SEE CASE 18 *)
( 59)   23              ANO:=0;                     (* SET NO ARGUMENTS YET.  *)
( 59)   24              G^.PNO[GTOP]:=S.PNO[SROW]; (* COPY THE PNO INTO GRAPH *)
( 59)   25              G^.DUMNUM[GTOP]:=SROW;      (* SET DUMNUM TO THE FAMILY NAME ENTRY *)
( 59)   26              G^.VBL[GTOP]:=FALSE;        (* IT IS NOT A VARIABLE *)
( 59)   27              G^.ORDIRR[GTOP]:=FALSE;     (* ORDER MATTERS *)
( 59)   28              G^.VAL[GTOP]:=[0..MNVAL];  (* THE VALUE IS INITIALLY * *)
( 59)   29              IF CHRR='E' THEN            (* IF WE ARE DOING DOMAIN STRUCTURES *)
( 59)   30              S.VTYPE[G^.PNO[GTOP]]:=3;  (* FORCE THIS DESCRIPTOR TO BE STRUCTURES *)
( 59)   31              END;(* DIGIT *)
( 59)   32
( 59)   33  16:       BEGIN                         (* A NUMBER HAS BEEN ENCOUNTERED *)
( 59)   34                (* PUSH DIGIT ON STK *)
( 59)   35
( 59)   36              FTOP := FTOP+1;             (* PUSH THE NEGATIVE OF NUMBER ONTO STACK *)
( 59)   37              FSTK[FTOP]:=-SROW;
( 59)   38              IF SROW>S.EVAL[G^.PNO[FSTK[1]]] THEN(* ADJUST THE DOMAIN DESCRIPTION *)
( 59)   39              S.EVAL[G^.PNO[FSTK[1]]]:=SROW;(* TO GO AT LEAST THIS HIGH *)
( 59)   40              IF CHRR<>'E' THEN           (* IF WE'RE DOING STRUCTURE INPUT...*)
( 59)   41                 IF SROW>S.NVAL[G^.PNO[FSTK[1]]] THEN(* AND SROW IS THE BIGGEST SO FAR *)
( 59)   42                 S.NVAL[G^.PNO[FSTK[1]]]:=SROW;(* THEN FORCE THE NUMBER OF VALUES TO SROW *)
( 59)   43              IF SROW<S.MVAL[G^.PNO[FSTK[1]]] THEN(* IF SROW IS TOO SMALL...*)
( 59)   44              S.MVAL[G^.PNO[FSTK[1]]]:=SROW;(* FORCE THE MIN VALUE TO SROW *)
( 59)   45              END;
( 59)   46  33:       BEGIN      (* HANDLE SYMBOLIC VALUE IN REFERENCE *)
( 59)   47              I:=G^.PNO[FSTK[1]];  (* GET SYMBOL TABLE INDEX OF SELECTOR *)
( 59)   48              J:=0;      (* VALUE TO ASSIGN *)
( 59)   49              WHILE (S.DPNO[I]<>0) AND (S.DPNO[I]<>SROW) DO BEGIN
( 59)   50                 I:=S.DPNO[I];    (* FOLLOW SYMBOL LINKS *)
( 59)   51                 J:=J+1;          (* INCREMENT VALUE COUNTER *)
( 59)   52                 END;(* WHILE *)
( 59)   53             (* I POINTS TO CURRENT END OF VALUE LIST *)
( 59)   54             (* J HAS VALUE TO ASSIGN *)
( 59)   55              IF S.DPNO[I]<>SROW THEN BEGIN  (* ADD SYMBOL TO LIST *)
( 59)   56                 S.DPNO[I]:=SROW;    (* LINK CURRENT END TO OUR VALUE *)
( 59)   57                 S.DPNO[SROW]:=0;    (* FORWARD LINK IS ZERO *)
( 59)   58                 END;(* IF I<>SROW *)
( 59)   59              SROW:=J;              (* UPDATE SROW *)
( 59)   60              FTOP:=FTOP+1;         (* PUSH VALUE ONTO FSTK *)
( 59)   61              FSTK[FTOP]:=-SROW;
( 59)   62              I:=G^.PNO[FSTK[1]]; (* GET SYMBOL TABLE INDEX OF SELECTOR AGAIN *)
( 59)   63              IF SROW>S.EVAL[I] THEN S.EVAL[I]:=SROW;
( 59)   64              IF CHRR<>'E' THEN
```

```
( 59)   65              IF SROW>S.NVAL[I] THEN S.NVAL[I]:=SROW;
( 59)   66            IF SROW<S.MVAL[I] THEN S.MVAL[I]:=SROW;
( 59)   67            END;(* IF X MODE *)
( 59)   68         END;(*CASE*)
( 59)   69
( 59)   70      END;
( 60)    1  *DECK PARSEARITH
( 60)    2  PROCEDURE PARSEARITH;                    (* PERFORM ARITH EXPRESSION PARSING SEMANTICS *)
( 60)    3  VAR
( 60)    4        I              : INTEGER;
( 60)    5
( 60)    6   (*               I N D U C E - 1  --  VLINT --  PROCESS  --  PARSEARITH  --  APUSH
( 60)    7   *)
( 60)    8  *CALL APUSH
( 60)    9   (*               I N D U C E - 1  --  VLINT  --  PROCESS  --  PARSEARITH  --  OPUSH
( 60)   10   *)
( 60)   11  *CALL OPUSH
( 60)   12
( 60)   13   (*1
( 60)   14                 I N D U C E - 1  --  VLINT  --  PROCESS  --  PARSEARITH(MAIN CODE)
( 60)   15   *)
( 60)   16  BEGIN
( 60)   17        WITH A^ DO BEGIN
( 60)   18           CASE PTBL.SRULE[-PROD] OF
( 60)   19
( 60)   20  21:         BEGIN                       (* PUSH FUNCTION SYMBOL *)
( 60)   21                 APUSH(FUNCT,S.PNO[SROW]);      STACK[SIZE].DUMMY[1]:=0;(* CLEAN DUMMY LIST *)
( 60)   22                 END;(* FUNCTION SYMBOL *)
( 60)   23
( 60)   24  22:         BEGIN                       (* ADD DUMMY VAR TO FUNCTION SYMBOL *)
( 60)   25                 I:=1;
( 60)   26                 WHILE (STACK[SIZE].DUMMY[I]<>0) AND (I<=MLNK) DO I:=I+1;(* FIND AN EMPTY DUMMY ENTRY *)
( 60)   27                 IF I>MLNK THEN WRITELN(OUTPUT,'FUNCTION HAS TOO MANY ARGUMENTS' ,' (MLNK=',MLNK:3,')');
( 60)   28                 STACK[SIZE].DUMMY[I]:=SROW;
( 60)   29                 END;(* DUMMY VAR *)
( 60)   30
( 60)   31  23:         APUSH(NUMBER,SROW);         (* A NUMBER *)
( 60)   32
( 60)   33  24:         OPUSH(MULTIPLY);            (* * OPERATOR *)
( 60)   34
( 60)   35  25:         BEGIN                       (* POP OPERATOR OFF OPSTK ONTO ARITHSTACK *)
( 60)   36                 APUSH(OPSTK[OTOP],0);      OTOP:=OTOP-1;
( 60)   37                 END;
( 60)   38
( 60)   39  26:         OPUSH(DIVIDE);              (* / OPERATOR *)
( 60)   40
( 60)   41  27:         OPUSH(ADD);                 (* + OPERATOR *)
( 60)   42
( 60)   43  28:         APUSH(MINUS,0);             (* PUSH UNARY MINUS ONTO ARITHSTACK *)
( 60)   44
( 60)   45  29:         OPUSH(SUBTRACT);            (* - OPERATOR *)
( 60)   46
( 60)   47  30:         OPUSH(MODULO);              (* % (MODULO DIVISION) *)
( 60)   48
( 60)   49  31:         ;                           (* NO OPERATION *)
( 60)   50
( 60)   51  32:         BEGIN(* INDICATE END OF PARSED EXPRESSION *)
( 60)   52                 DONE:=TRUE;      S.EVAL[STACK[1].ARGUMENT]:=MNVAL;(* SET UP SYMBOL TABLE ENTRIES *)
```

```
( 60)   53                S.NVAL[STACK[1].ARGUMENT]:=MNVAL;
( 60)   54                S.MVAL[STACK[1].ARGUMENT]:=0;
( 60)   55                END;(*END OF EXPRESSION *)
( 60)   56             END;(* CASE *)
( 60)   57          END;(* WITH *)
( 60)   58       END;(* PARSE ARITH *)
( 61)    1  *DECK APUSH
( 61)    2  PROCEDURE APUSH
( 61)    3       (ACT             : ACTIONCODE;
( 61)    4        ARG             : INTEGER);
( 61)    5  BEGIN
( 61)    6       WITH A^ DO BEGIN
( 61)    7          IF SIZE>=MAXASTACK THEN WRITELN(OUTPUT,'ARITHMETIC EXPRESSION TOO LARGE', ' (MAXASTACK =',MAXASTACK:3,')');
( 61)    8          SIZE:=SIZE+1;       STACK[SIZE].ACTION:=ACT;       STACK[SIZE].ARGUMENT:=ARG;
( 61)    9          END;(* WITH *)
( 61)   10       END;(*APUSH*)
( 62)    1  *DECK OPUSH
( 62)    2  PROCEDURE OPUSH
( 62)    3       (ACT             : ACTIONCODE);     (* PUSH ACTION CODE ONTO OPSTK *)
( 62)    4  BEGIN
( 62)    5       OTOP := OTOP +1;       OPSTK[OTOP]:=ACT;
( 62)    6       END;(*OPUSH*)
( 63)    1  *DECK COSTG
( 63)    2  PROCEDURE COSTG
( 63)    3       (P               : GPTR;
( 63)    4        CT              : INTEGER);
( 63)    5  LABEL
( 63)    6       6;
( 63)    7  VAR
( 63)    8       J,I              : INTEGER;
( 63)    9       INSD,CTNEG       : BOOLEAN;
( 63)   10       Q                : GPTR;
( 63)   11  BEGIN(*COSTG*)
( 63)   12
( 63)   13       IF CT<0 THEN                        (* CT IS COST FUNCTION #.  IF NEG WE NEGATE THE ANSWER *)
( 63)   14       CTNEG:=TRUE
( 63)   15       ELSE                                (* REMEMBER THE SIGN OF CT *)
( 63)   16       CTNEG:=FALSE;       CT:=ABS(CT);    (* AND THEN TAKE IT'S ABSOLUTE VALUE *)
( 63)   17       IF CT IN [1,2,3,4,5] THEN           (* IS IT LEGAL?? *)
( 63)   18       CASE CT OF
( 63)   19  1,3:     BEGIN
( 63)   20              CASE CT OF
( 63)   21  1:             INSD:=TRUE;              (* NUMBER OF CFORMULAS IN F1 COVERED,  SET "COVER" *)
( 63)   22  3:             INSD:=FALSE              (* NUMBER OF CFORMULAS IN FO INTERSECTED, SET "INTERSECT" *)
( 63)   23              END;(*CASE STMT*)
( 63)   24
( 63)   25              P^.COST[CT]:=0;             (* INIT COST TO 0 *)
( 63)   26              Q:=GSET;
( 63)   27              IF (CT=1) OR ((CT=3) AND (PRM.DESCTYPE=DISCRIMINANT)) THEN WHILE (Q<>NIL) DO
( 63)   28                 BEGIN(* CHECK OUT ALL OF THE RULES IN RULEBASE *)
( 63)   29                    IF (CT=1)AND(ES IN Q^.ESET)AND(Q^.FP)(* IF ITS IN F1 AND CT=1 *)
( 63)   30                    OR( CT=3)AND (NOT (ES IN Q^.ESET)) THEN(* OR IN FO AND CT=3 *)
( 63)   31                       IF SUBG1(P,Q,0,AQP.FREEC,TRUE,NIL) THEN(* AND IT COVERS/INTERSECTS *)
( 63)   32                       P^.COST[CT]:=P^.COST[CT]+1;(* COUNT IT IN THE COST *)
( 63)   33                    Q:=Q^.NXTN;           (* AND EXAMINE NEXT RULE *)
( 63)   34                    END;(*WHILE Q<>NIL*)
( 63)   35
( 63)   36              END;(*CASE 1*)
```

```
( 63)   37
( 63)   38  2,4:     BEGIN                          (* 2=# OF SELECTORS, 4=TOTAL COST OF ALL VARS W/ REF <> * *)
( 63)   39              P^.COST[CT]:=0;             (* INIT COST TO 0 *)
( 63)   40              FOR J:=1 TO GSIZE DO        (* FOR EACH DESCRIPTOR ... *)
( 63)   41              IF P^.LNK[J,1]<>0 THEN      (* CHECK EACH LINK... *)
( 63)   42                 IF NOT P^.VBL[J] THEN    (* THAT IS A FUNCTION FOR PREDICATE ... *)
( 63)   43                    IF (S.NARG[ABS(P^.PNO[J])]>1)OR(P^.VAL[J]<>[0..MNVAL]) THEN(* WITH >1 ARG OR REFERENCE <> * *)
( 63)   44                       IF CT=2 THEN P^.COST[2]:=P^.COST[2]+1(* COUNT THE SELECTOR *)
( 63)   45                       ELSE P^.COST[4]:=P^.COST[4]+S.VCOST[ABS(P^.PNO[J])];(* ADD IN THE COST *)
( 63)   46              IF P^.MSEL<>NIL THEN        (* NOW CHECK THE META SELECTORS *)
( 63)   47              FOR J:=1 TO MST.NMST DO     (* FOR EACH MS *)
( 63)   48                 IF P^.MSEL^.CVAL[MST.PTR[J]]<>[0..MNVAL] THEN(* IF ITS <> *  *)
( 63)   49                    IF CT=2 THEN P^.COST[2]:=P^.COST[2]+1(* COUNT AS A SELECTOR *)
( 63)   50                    ELSE P^.COST[4]:=P^.COST[4]+ S.VCOST[ABS(MST.PNO[MST.PTR[J]])];(* OR ADD IN THE
( 63)   51                          COST OF ITS ASSOCIATED FUNCTION *)
( 63)   52              END;(*CASE 2*)
( 63)   53  5:       BEGIN                 (*  SUM COSTS OF ALL VARIABLES INVOLVED *)
( 63)   54              P^.COST[CT]:=0; (*  INIT COST TO ZERO *)
( 63)   55              FOR J:=1 TO GSIZE DO IF P^.LNK[J,1]<>0 THEN
( 63)   56                 IF NOT P^.VBL[J] THEN    (* IF THIS DESCRIPTOR IS A FUNCTION... *)
( 63)   57                 BEGIN
( 63)   58                    I:=1;          (* SCAN ALL LINKS,  ADD UP ALL COSTS *)
( 63)   59                    WHILE (P^.LNK[J,I]<>0) AND (I<=MLNK) DO BEGIN
( 63)   60                       IF P^.VBL[P^.LNK[J,I]] THEN P^.COST[CT]:= P^.COST[CT] + S.VCOST[P^.DPNO[P^.LNK[J,I]]];
( 63)   61                       I:= I+ 1;    (* INCREMENT I *)
( 63)   62                       END;(* WHILE *)
( 63)   63                    END;(* IF NOT VBL *)
( 63)   64              END;(* CASE 5 *)
( 63)   65
( 63)   66           END;(*CASE STMT*)
( 63)   67
( 63)   68        IF CTNEG THEN                    (* NOW FIX SIGN OF THE COST *)
( 63)   69        P^.COST[CT]:=-P^.COST[CT];
( 63)   70        END; (* COSTG  *)
( 64)    1  *DECK ADDTOMQ
( 64)    2  PROCEDURE ADDTOMQ
( 64)    3       (G               : GPTR);(* PURPOSE:  TO MAKE A COPY OF THIS GRAPH AND PUT IT ON MQ *)
( 64)    4  VAR
( 64)    5        G1              : GPTR;
( 64)    6  BEGIN
( 64)    7        NEWG(G1);                        (* GET A NEW GRAPH *)
( 64)    8        G1^:=G^;                         (* COPY THE GRAPH STRUCTURE COMPLETELY *)
( 64)    9        G^.RNO := CRULENO-1;             (* GIVE THE OLD GRAPH THE NEW RULE NUMBER *)
( 64)   10        G1^.NXTN := MQ;                  (* PLACE G1 AT HEAD OF MQ *)
( 64)   11        MQ:=G1;              NMQ := NMQ+1; (* COUNT IT IN NMQ *)
( 64)   12        END;(*ADDTOMQ*)
( 65)    1  *DECK TRIMG
( 65)    2  PROCEDURE TRIMG
( 65)    3       (VAR NSTAR       : GPTR;          (* NSTAR: LIST OF GRAPHS TO TRIM *)
( 65)    4        MAXS            : INTEGER);      (* MAXS: MAXSTAR PARAMETER *)
( 65)    5  LABEL
( 65)    6        1,
( 65)    7        2,
( 65)    8        99;
( 65)    9  TYPE
( 65)   10        ATYPE           = ARRAY[0..300] OF GPTR;
( 65)   11  VAR
( 65)   12        CA              : ATYPE;
```

```
( 65)   13        X               : REAL;
( 65)   14        P,Q             : GPTR;
( 65)   15        NC,I,J,IB,IC    : INTEGER;
( 65)   16  BEGIN(*TRIMG*)
( 65)   17
( 65)   18        IC:=1;      IB:=1;      P:=NSTAR;      NC:=0;
( 65)   19        WHILE P<>NIL DO                  (* LOOP THRU THE STAR--PUT CONSISTENT RULES ON MQ *)
( 65)   20        BEGIN
( 65)   21           Q:=P;
( 65)   22           IF P^.FP THEN                 (* IF ITS "IN" *)
( 65)   23           BEGIN
( 65)   24              IF (P^.COST[3]=0) AND (PRM.DESCTYPE=DISCRIMINANT) THEN ADDTOMQ(P);(* MAKE A COPY ON MQ *)
( 65)   25              NC:=NC+1;      CA[NC]:=P;  (* PUT POINTER ON THE CANDIDATE ARRAY LIST *)
( 65)   26              P:=P^.NXTN;
( 65)   27              END(*IF THEN*)
( 65)   28
( 65)   29           ELSE(*FP FALSE*)
( 65)   30
( 65)   31           BEGIN
( 65)   32              IF P^.COST[3]=100 THEN     (* IF NOT ON FP AND COST=100, RELEASE IT???????*)
( 65)   33              BEGIN
( 65)   34                 P^.MSEL^.NXTC:=AQP.FREEC;      AQP.FREEC:=P^.MSEL;      P^.MSEL:=NIL;
( 65)   35                 END;
( 65)   36              P:=P^.NXTN;      Q^.NXTN:=FREEG;      FREEG:=Q;
( 65)   37              END;
( 65)   38           END;(*WHILE P<>NIL *)
( 65)   39
( 65)   40        CA[NC+1]:=CA[NC];                (* COPY ENDS OF ARRAY FOR THIS ALGORITHM *)
( 65)   41        CA[0]:=CA[1];
( 65)   42  1:    IF NC<=MAXS THEN                 (* IF NOTHING TO TRIM, QUIT *)
( 65)   43        GOTO 99;
( 65)   44        I:=1;
( 65)   45        IF MAXS=0 THEN GOTO 2;
( 65)   46        FOR I:=IB TO NC-IB DO            (* BUBBLE SORT THE POINTER ARRAY ACCORDING TO COST IC *)
( 65)   47        FOR J:=I+IB TO NC DO IF CA[J]^.COST[ABS(PRM.CSTF[IC])] < CA[I]^.COST[ABS(PRM.CSTF[IC])] THEN BEGIN
( 65)   48           P:=CA[J];      CA[J]:=CA[I];      CA[I]:=P;
( 65)   49           END;
( 65)   50        I:=MAXS+1;
( 65)   51        IF PRM.TOLER[IC]=TRUNC(PRM.TOLER[IC]) THEN(* CONVERT RELATIVE TOLERANCES TO ABSOLUTE *)
( 65)   52        X:=PRM.TOLER[IC]
( 65)   53        ELSE X:=PRM.TOLER[IC]*(CA[NC]^.COST[ABS(PRM.CSTF [IC])]-CA[1]^.COST[ABS(PRM.CSTF [IC])]) ;
( 65)   54        IF IC<>PRM.NF THEN               (* IF THIS ISN'T THE LAST COST FUNCTION *)
( 65)   55        WHILE (CA[MAXS]^.COST[ABS(PRM.CSTF[IC])] >= CA[I]^.COST[ABS(PRM.CSTF[IC])] - X) AND (I<=NC) DO
( 65)   56           I:=I+1;(* ADVANCE THE POINTER UNTIL WE LEAVE THE TOLERANCE*)
( 65)   57        (* RETURN ELEMENTS FROM I TO NC*)
( 65)   58
( 65)   59  2:    FOR J:=I TO NC DO BEGIN          (* RELEASE GRAPHS BELOW THE TRIM POINT *)
( 65)   60           CA[J]^.NXTN:=FREEG;      FREEG:=CA[J];
( 65)   61           END;
( 65)   62        NC:=I-1;      IB:=MAXS-1;
( 65)   63        WHILE (CA[MAXS]^.COST[ABS(PRM.CSTF[IC])] <= CA [IB]^.COST[ABS(PRM.CSTF[IC])] + X) AND (IB>0) DO IB:=IB-1;
( 65)   64        IB:=IB+1;      IC:=IC+1;
( 65)   65        IF IC<=PRM.NF THEN               (* IF MORE TO DO, CONTINUE *)
( 65)   66        GOTO 1;
( 65)   67  99:   NSTAR:=NIL;
( 65)   68        FOR I:=1 TO NC DO                (* RETURN THE STAR AS TRIMMED *)
( 65)   69        BEGIN
( 65)   70           CA[I]^.NXTN:=NSTAR;      NSTAR:=CA[I];
```

```
( 65)   71          END;
( 65)   72        END;(*TRIMG*)
( 66)    1  *DECK COMPMS
( 66)    2  PROCEDURE COMPMS
( 66)    3       (GSET            : GPTR;            (* COMPUTE METASELECTORS *)
( 66)    4        ES,MPNO,MTVAL   : INTEGER;
( 66)    5        VAR NPT1        : IARRAY;           (* NUMBER OF PARTS ARRAYS *)
( 66)    6        VAR NPTO        : IARRAY;          (* AND FOR FO *)
( 66)    7        VAR FA1         : INTEGER;         (* FORALL, F1COV *)
( 66)    8        VAR FAO         : INTEGER);        (* FORALL FO COVER *)
( 66)    9  VAR
( 66)   10        G               : GPTR;
( 66)   11        I,J,K,L,M,N     : INTEGER;
( 66)   12  BEGIN(* INPUT LIST OF EVENTS,  FNCTNS AND VAUES.  CALCULATE
( 66)   13         META SELECTORS NPT AND FORALL; ADD TO EVENT*)
( 66)   14
( 66)   15
( 66)   16      (* ADD INFO TO MST *)
( 66)   17
( 66)   18        FOR I:=1 TO 2 DO BEGIN           (* UPDATE THE MST TABLE WITH *)
( 66)   19           MST.PNO[MST.PTR[MST.NMST+I]]:=MPNO;(* DEFINITIONS FOR THIS MS *)
( 66)   20           MST.VAL[MST.PTR[MST.NMST+I]]:=MTVAL;(* POINT TO FORALL(1) AND #PT (2) *)
( 66)   21           MST.SYMPTR[MST.PTR[MST.NMST+I]]:=I;
( 66)   22           END;
( 66)   23        MST.NMST:=MST.NMST+2;       N:=MST.NMST;       G:=GSET;       NF1:=0;
( 66)   24        WHILE G<>NIL DO                  (* EXAMINE EACH RULE IN RULE BASE *)
( 66)   25        BEGIN
( 66)   26           K:=0;       L:=0;
( 66)   27           FOR I:=1 TO GSIZE DO          (* EXAMINE THIS RULE *)
( 66)   28           IF (G^.PNO[I]=MST.PNO[MST.PTR[N]]) AND (G^.LNK[I,1]<>0) THEN(* IS THIS DESCRIPTOR THE ONE WE ARE DOING? *)
( 66)   29           BEGIN
( 66)   30              K:=K+1;                    (* YES, COUNT TOTAL # OF OCCURANCES OF THIS DESCRIPTOR *)
( 66)   31              MST.VARPTR[MST.PTR[N]]:=G^.PNO[G^.LNK[I,1]];  (*  REMEMBER THE FAMILY
( 66)   32                 NAME OF THE DUMMY VARIABLE *)
( 66)   33              MST.VARPTR[MST.PTR[N-1]]:=G^.PNO[G^.LNK[I,1]];(*THE ABOVE 2 STATEMENTS PUT THE DUMMY VAR OF THIS FUNCTION
( 66)   34                    INTO THE MST FOR USE BY THE FORALL AND #PT PRINT
( 66)   35                    ROUTINES.  NOTE THAT IF THE FUNCTION USES TWO DIFFERENT
( 66)   36                    CLASSES OF DUMMIES, ONLY THE LAST ONE WILL APPEAR *)
( 66)   37              IF MST.VAL[MST.PTR[N]] IN G^.VAL[I] THEN  (* IS THE VALUE WE'RE DOING IN HIS VALUES? *)
( 66)   38              BEGIN
( 66)   39                 L:=L+1;                 (* COUNT THE NUMBER OF SELECTORS COVERED *)
( 66)   40                 END;
( 66)   41              END;
( 66)   42           G^.MSEL^.CVAL[MST.PTR[N]]:=[L];(* THAT NUMBER (L) IS THE #PT VALUE *)
( 66)   43           IF L>S.NVAL[MST.SYMPTR[MST.PTR[N]]] THEN(* FIX THE DOMAIN DEFN IF NECESSARY *)
( 66)   44           BEGIN
( 66)   45              S.NVAL[MST.SYMPTR[MST.PTR[N]]]:=L;      S.EVAL[MST.SYMPTR[MST.PTR[N]]]:=L;
( 66)   46              END;
( 66)   47           IF (K=L) AND (K>0) THEN                (* IF K(TOTAL NUMBER OF SELECTORS) = L(NUMBER ACTUALLY COVERED) *)
( 66)   48           G^.MSEL^.CVAL[MST.PTR[N-1]]:=[1] (* THE WE HAVE A "FORALL" PREDICATE *)
( 66)   49           ELSE G^.MSEL^.CVAL[MST.PTR[N-1]]:=[0];(* ELSE FORALL IS FALSE *)
( 66)   50           IF (K=L) AND (K>0) THEN
( 66)   51              IF ES IN G^.ESET THEN    (* IF THIS RULE IS IN F1 AND FORALL, *)
( 66)   52              FA1:=FA1+1                 (* ADD 1 TO F1COV *)
( 66)   53              ELSE FAO:=FAO+1;           (* ADD 1 TO FOCOV *)
( 66)   54           IF ES IN G^.ESET THEN         (* IF THIS IS IN F1... *)
( 66)   55           NF1:=NF1+1;                   (* ADD 1 TO THE NUMBER OF F1 RULES *)
( 66)   56           IF ES IN G^.ESET THEN         (* PROCESS THE #PT TO FIGURE F1/FO COV *)
```

```
( 66)   57          NPT1[L]:=NPT1[L]+1              (* IF IN F1, ADD 1 TO THE APPROPRIATE VALUE ENTRY *)
( 66)   58          ELSE NPTO[L]:=NPTO[L]+1;        (* ELSE DO THE SAME FOR FO ENTRY *)
( 66)   59          G:=G^.NXTN;                     (* DO NEXT RULE *)
( 66)   60          END;(*WHILE*)
( 66)   61
( 66)   62        END;(* COMPMS *)
( 67)    1  *DECK TRIMM
( 67)    2  PROCEDURE TRIMM;                       (* TRIM THE METASELECTOR TABLE TO METATRIM *)
( 67)    3  VAR
( 67)    4        I,J,K,L,M       : INTEGER;
( 67)    5        G               : GPTR;
( 67)    6  BEGIN
( 67)    7        IF MST.METATRIM<MST.NMST THEN    (* IF METATRIM IS < NMST, WE HAVE TO SORT FIRST *)
( 67)    8        BEGIN
( 67)    9           FOR I:=1 TO MST.METATRIM DO   (* BUBBLE SORT BY ASCENDING F1COV, DESCENDING FOCOV *)
( 67)   10           FOR J:=I+1 TO MST.NMST DO
( 67)   11              IF (MST.F1COV[MST.PTR[I]]<MST.F1COV [MST.PTR[J]])OR (MST.F1COV[MST.PTR[I ]]=MST.F1COV[MST.PTR[J]])AND
( 67)   12                 (MST.FOCOV [MST.PTR[I]]>MST.FOCOV[MST.PTR[J]]) THEN BEGIN(* SWITCH EM *)
( 67)   13                 L:=MST.PTR[I];      MST.PTR[I]:=MST.PTR[J];      MST.PTR[J]:=L;
( 67)   14                 END;
( 67)   15           MST.NMST:=MST.METATRIM;       (* ALL DONE, NOW TRIM THE LIST *)
( 67)   16           END;
( 67)   17        END;(*  TRIMM *)
( 68)    1  *DECK ADDMETA
( 68)    2  PROCEDURE ADDMETA;(* THIS PROCEDURE CALCULATES A SET OF META SELECTORS
( 68)    3    AND HAS THEM LOADED IN TO THE EVENT *)
( 68)    4
( 68)    5  LABEL
( 68)    6        10;
( 68)    7
( 68)    8  VAR
( 68)    9        I,J,K,L,FA1,FAO: INTEGER;
( 68)   10        NPT1,NPTO      : IARRAY;
( 68)   11  BEGIN
( 68)   12        FOR I:=1 TO SYMSZE DO IF (S.NARG[I]=1)AND(S.NAME[I,4]<>'-') THEN(* DONT APPLY MS TO MST- AND LST- PREDICATES *)
( 68)   13        FOR J:=S.MVAL[I] TO S.NVAL[I] DO  (* FOR EACH VALUE IN THE DOMAIN *)
( 68)   14        BEGIN
( 68)   15           FOR L:=0 TO MNVAL DO          (* CLEAN THE NPT ARRAYS *)
( 68)   16           BEGIN
( 68)   17              NPT1[L]:=0;      NPTO[L]:=0;
( 68)   18              END;
( 68)   19           FA1:=0;                       (* CLEAN THE FORALL VALUES *)
( 68)   20           FAO:=0;
( 68)   21           IF MST.NMST+2>GSIZE THEN BEGIN     (* WILL THESE GUYS FIT?? *)
( 68)   22              WRITELN('MST EXCEEDED--ONLY GSIZE MSS COMPUTED');
( 68)   23              GOTO 10;
( 68)   24              END;
( 68)   25
( 68)   26     (*  FOR EACH UNARY PREDICATE (THAT ISN'T A MST- OR LST- )
( 68)   27         AND FOR EACH VALUE IN THE DOMAIN OF THE PREDICATE,
( 68)   28         WE CALCULATE A FORALL AND A #PT METASELECTOR.
( 68)   29         WE ADD 2 ROWS TO THE MST AND THEN GO THRU ALL OF THE
( 68)   30         RULES AND COMPUTE THE F1COV AND FOCOV VALUES.
( 68)   31         FOR "FORALL", THIS IS EASY.  F1COV IS THE NUMBER
( 68)   32         OF F1 RULES FOR WHICH IT IS TRUE, FOCOV IS THE NUMBER
( 68)   33         OF FO RULES FOR WHICH IT IS TRUE.
( 68)   34         FOR #PT, HOWEVER, THIS IS TRICKY.  WE DON'T KNOW WHAT THE
( 68)   35         VALUE OF THE #PT SELECTOR SHOULD BE, SO WE BUILD AN ARRAY
```

```
( 68)   36        (NPT1 AND NPT0) INDEXED BY POSSIBLE VALUES OF THE #PT SELECTOR.
( 68)   37        AFTER WE'RE DONE, WE CHOOSE THE VALUE WHICH GIVES THE BIGGEST
( 68)   38        F1COV AND MAKE IT THE VALUE OF THE SELECTOR AND THE VALUE OF
( 68)   39        THE F1COV/FOCOV ENTRY.*)
( 68)   40
( 68)   41           COMPMS(GSET,ES,I,J,NPT1,NPT0,FA1,FA0);    (* COMPUTE F1/F0 COV ETC *)
( 68)   42           MST.F1COV[MST.PTR[MST.NMST-1]]:=FA1;    (* GET THE "FORALL" F1COV *)
( 68)   43           MST.FOCOV[MST.PTR[MST.NMST-1]]:=FA0;    (* AND THE "FORALL" FOCOV *)
( 68)   44           K:=-1000;                        (* FIGURE MAX F1COV FOR #PT, START W/ MAX OF -1000 *)
( 68)   45           FOR L:=0 TO MNVAL DO IF NPT1[L]>K THEN(* FOR EACH POSSIBLE VALUE, IF THE F1COV > MAX *)
( 68)   46           BEGIN
( 68)   47              K:=NPT1[L];                   (* MAKE IT THE MAX *)
( 68)   48              FA1:=L;                       (* AND REMEMBER WHICH ONE IT IS *)
( 68)   49              END;
( 68)   50           MST.F1COV[MST.PTR[MST.NMST]]:=K;    (* PUT THE F1COV VALUE INTO THE MST *)
( 68)   51           MST.FOCOV[MST.PTR[MST.NMST]]:=NPT0[FA1];    (* AND TAKE THE CORRESPONDING FO COV *)
( 68)   52           END;
( 68)   53  10:    TRIMM;                             (* TRIMM THE MST TO METATRIM ELEMENTS *)
( 68)   54        IF 6 IN TRACE THEN BEGIN
( 68)   55           EXPLN(6);      PMETAD;           (* PRINT THE MST *)
( 68)   56           END;(*IF TRACE*)
( 68)   57
( 68)   58        END;(* ADDMETA *)
( 69)    1  *DECK PROCARITH
( 69)    2  PROCEDURE PROCARITH
( 69)    3        (A                : ARITHPTR;
( 69)    4         G                : GPTR);(* PURPOSE:  TO LOOP THRU ALL ELEMENTS OF THE G LIST (GSET)
( 69)    5       ATTEMPTING TO ADD THE ARITHMETIC DERIVED DESCRIPTOR IN A^ TO EACH
( 69)    6       GRAPH IN TURN.  THIS IS ACCOMPLISHED BY FIRST CREATING A TEMP GRAPH
( 69)    7       (TOFIND) WHICH CONTAINS THE NECESSARY FUNCTIONS AND DUMMIES FROM
( 69)    8       THE RIGHT HAND SIDE OF THE ARITHDD.  THEN WE CALL SUBG1 TO FIND
( 69)    9       ALL ISOMORPHISMS BETWEEN TOFIND AND THE RULES IN GSET.  $UBG1 CALLS
( 69)   10       CALCARITH WITH ACTUALLY PERFORMS THE INSERTIONS INTO THE RULES.
( 69)   11     *)
( 69)   12  VAR
( 69)   13        I,J               : INTEGER;(* MISC INTEGERS *)
( 69)   14        FROW              : INTEGER;(* ROW OF FUNCTION NODE IN TOFIND *)
( 69)   15        DROW              : INTEGER;(* ROW OF DUMMY NODE IN TOFIND *)
( 69)   16        TOFIND            : GPTR;(* TEMP GRAPH CONTAINING RHS OF ARITHDD EQUATION *)
( 69)   17        GX                : 1..GSIZE;
( 69)   18        TEMPG             : GPTR;(* TEMP GRAPH PTR *)
( 69)   19     (*              I N D U C E - 1  --  PROCARITH  --  BUILDG
( 69)   20     *)
( 69)   21  *CALL BUILDG
( 69)   22     (*1
( 69)   23                  I N D U C E - 1  --  PROCARITH (MAIN CODE)
( 69)   24     *)
( 69)   25  BEGIN
( 69)   26        IF NOT A^.APPLIED THEN BEGIN
( 69)   27           NEWG(TOFIND);                  (* GET A TEMP GRAPH FOR BUILDG *)
( 69)   28           WITH TOFIND^ DO BEGIN
( 69)   29              FOR GX:=1 TO GSIZE DO PNO[GX] := 0;
( 69)   30              END;
( 69)   31           BUILDG(TOFIND,A);              (* SET UP RHS GRAPH HERE *)
( 69)   32           TEMPG:=G;                      (* NOW LOOP THRU GSET *)
( 69)   33           WHILE TEMPG<>NIL DO BEGIN
( 69)   34              IF SUBG1(TOFIND,TEMPG,3,AQP.FREEC,TRUE,A) THEN ;
( 69)   35              TEMPG:=TEMPG^.NXTN;
```

```
( 69)   36              END;(* WHILE *)
( 69)   37                                          (* SUBG1 :  FIND ALL SUBGRAPHS OF TEMPG ISOMORPHIC TO "TOFIND" *)
( 69)   38                                          (* CALL CALCARITH TO PROCESS EACH ONE *)
( 69)   39                                          (* NIL=> NO COMPLEXES TO BUILD *)
( 69)   40                                          (* TRUE=> REFERENCES SHOULD COVER (ALL OF OUR REFS ARE * *)
( 69)   41                                          (* A=> PASSED TO CALCARITH *)
( 69)   42           A^.APPLIED:=TRUE;
( 69)   43           END;(* IF NOT APPLIED *)
( 69)   44       END;(* PROCARITH *)
( 70)    1  *DECK BUILDG
( 70)    2  PROCEDURE BUILDG
( 70)    3       (TOFIND          : GPTR;
( 70)    4        A               : ARITHPTR);(* PURPOSE:  TO BUILD THE GRAPH "TOFIND" FROM THE ARITHSTACK
( 70)    5   *)
( 70)    6  VAR
( 70)    7        I               : INTEGER;(*               I N D U C E - 1  --  PROCARITH  --  BUILDG  --  ADDLINK
( 70)    8   *)
( 70)    9  *CALL ADDLINK
( 70)   10   (*1
( 70)   11                 I N D U C E - 1  --  PROCARITH  --  BUILDG  --  FINDFUNC
( 70)   12   *)
( 70)   13  *CALL FINDFUNC
( 70)   14   (*1
( 70)   15                 I N D U C E - 1  --  PROCARITH  --  BUILDG (MAIN CODE)
( 70)   16   *)
( 70)   17  BEGIN
( 70)   18       FOR I:=2 TO A^.SIZE DO BEGIN      (* DONT EXAMINE ROW 1 (LEFT HAND SIDE)*)
( 70)   19          IF A^.STACK[I].ACTION=FUNCT THEN BEGIN
( 70)   20             FINDFUNC(A^.STACK[I].ARGUMENT,FROW,FALSE);(* ADD THIS FUNCTION *)
( 70)   21             J:=1;                      (* PROCESS EACH DUMMY *)
( 70)   22             WHILE (A^.STACK[I].DUMMY[J]<>0) DO BEGIN
( 70)   23                FINDFUNC(A^.STACK[I].DUMMY[J],DROW,TRUE);(* FIND/ADD THIS DUMMY *)
( 70)   24                ADDLINK(DROW,FROW);     (* LINK DUMMY TO FUNCTION *)
( 70)   25                ADDLINK(FROW,DROW);     (* LINK FUNCTION TO DUMMY *)
( 70)   26                J:=J+1;
( 70)   27                END;(* WHILE DUMMY[J]<>0*)
( 70)   28             A^.STACK[I].ARGUMENT:=FROW;(* SAVE ROW NUMBER IN ARGUMENT *)
( 70)   29             END;(* IF ACTION=FUNCT *)
( 70)   30          END;(* FOR I *)
( 70)   31       END;(* BUILDG *)
( 71)    1  *DECK ADDLINK
( 71)    2  PROCEDURE ADDLINK
( 71)    3       (X1,X2           : INTEGER);(* PURPOSE:  TO ADD A LINK IN THE GRAPH "TOFIND" FROM X1 TO X2
( 71)    4   *)
( 71)    5  VAR
( 71)    6       I                : INTEGER;
( 71)    7  BEGIN
( 71)    8       I:=1;(* SEARCH THE X1 NODE FOR AN EMPTY LINK  *)
( 71)    9       WITH TOFIND^ DO BEGIN
( 71)   10          WHILE (LNK[X1,I]<>0) AND (I<=MLNK) DO I:=I+1;
( 71)   11          IF I>MLNK THEN WRITELN(OUTPUT,'ARITHMETIC EXPRESSION CANT FIT INTO GRAPH ', '(MLNK=',MLNK:3,')');
( 71)   12          LNK[X1,I]:=X2;
( 71)   13          END;(* WITH *)
( 71)   14       END;(* ADDLINK *)
( 72)    1  *DECK FINDFUNC
( 72)    2  PROCEDURE FINDFUNC
( 72)    3       (SX              : INTEGER;
( 72)    4        VAR GX          : INTEGER;
```

```
( 72)    5        VBLFLAG         : BOOLEAN);(* PURPOSE:  TO ADD NEW NODES IN THE GRAPH AND INITIALIZE THEM.  IF
( 72)    6      CALLED WITH VBLFLAG=TRUE, IT WILL FIRST ATTEMPT TO FIND AN OLD VARIABLE
( 72)    7      NODE BEFORE CREATING A NEW ONE.  WITH VBLFLAG=FALSE WE ARE PROCESSING
( 72)    8      A FUNCTION.  EACH FUNCTION SHOULD HAVE A SEPARATE SELECTOR.
( 72)    9    *)
( 72)   10    VAR
( 72)   11        FOUND           : BOOLEAN;
( 72)   12        ILINK           : 1..MLNK;
( 72)   13    BEGIN
( 72)   14        GX:=1;(* RETURNS GX AS NODE INDEX IT FOUND--START AT 1 *)
( 72)   15        FOUND := FALSE;
( 72)   16        WITH TOFIND^ DO BEGIN
( 72)   17           WHILE (GX<GSIZE) AND NOT FOUND DO BEGIN
( 72)   18              FOUND := ((DPNO[GX]=SX) AND VBLFLAG) OR (PNO[GX]=0);
( 72)   19              IF NOT FOUND THEN GX := GX + 1;
( 72)   20              END;
( 72)   21           IF GX>=GSIZE THEN WRITELN(OUTPUT,'ARITHMETIC EXPRESSION CANT FIT INTO GRAPH', ' (GSIZE=',GSIZE:3,')');
( 72)   22           IF PNO[GX]=0 THEN BEGIN(* IF PNO IS ZERO, WE DIDN'T FIND IT, ADD IT *)
( 72)   23              VBL[GX]:=VBLFLAG;(* CALLER TELLS US IF ITS A VARIABLE *)
( 72)   24              ORDIRR[GX]:=(S.NARG[SX]<=1);(* ASSUME ORDER IMPORTANT *)
( 72)   25              VAL[GX]:=[0..MNVAL];(* SET IT TO MATCH ANYTHING *)
( 72)   26              IF VBLFLAG THEN PNO[GX]:=S.DPNO[SX] ELSE PNO[GX]:=SX;
( 72)   27              COUNT[GX] := 1;       DPNO[GX]:=SX;
( 72)   28              FOR ILINK := 1 TO MLNK DO LNK[GX,ILINK] := 0;
( 72)   29              END;(* IF PNO=0*)
( 72)   30           END;(* WITH *)
( 72)   31        END;(*FINDFUNC*)
( 73)    1    *DECK ADDML
( 73)    2    PROCEDURE ADDML;
( 73)    3    LABEL
( 73)    4        2;(* SELECT ONE PREDICATE AND ADD LEFT AND RIGHT ENDS TO STABLE.
( 73)    5     THEN ADD THE LEFT OR RIGHT END FUNCTION TO THE GRAPH FOR EACH EVENT*)
( 73)    6
( 73)    7    VAR
( 73)    8        I,J,K,L,M       : INTEGER;(* MISC INTEGERS *)
( 73)    9        G               : GPTR;
( 73)   10    BEGIN
( 73)   11        FOR I:=1 TO S.NELT DO              (* FOR EACH BINARY PREDICATE *)
( 73)   12        IF (S.NARG[I]=2)AND(S.MVAL[I]=1) THEN BEGIN(*ADD TO STABLE*)
( 73)   13
( 73)   14           S.NELT:=S.NELT+2;               (* ADD MST- AND LST- ENTRIES TO SYMTAB *)
( 73)   15           S.NAME[S.NELT-1]:='MST-       ';
( 73)   16           S.NAME[S.NELT]:='LST-       ';
( 73)   17           FOR K:=S.NELT-1 TO S.NELT DO  (* SET UP THE OTHER INFORMATION *)
( 73)   18           BEGIN
( 73)   19              FOR J:=5 TO 10 DO S.NAME[K,J]:=S.NAME[I,J-4];(* COPY THE FIRST 6 CHARS OF THE ORIGINAL NAME *)
( 73)   20              S.PNO[K]:=K;                 (* POINT PNO AT ITSELF *)
( 73)   21              S.NARG[K]:=1;                (* 1 ARGUMENT *)
( 73)   22              S.NVAL[K]:=1;                (* ITS A PREDICATE, NVAL=MVAL=EVAL=1 *)
( 73)   23              S.MVAL[K]:=1;         S.EVAL[K]:=1;
( 73)   24              END;(*FOR K*)
( 73)   25
( 73)   26           G:=GSET;                        (* NOW GO THRU THE RULE BASE AND ADD TO APPROPRIATE RULES *)
( 73)   27           WHILE G<>NIL DO BEGIN
( 73)   28              FOR J:=1 TO GSIZE DO IF G^.PNO[J]=I THEN(* IF THIS NODE IS OUR GUY *)
( 73)   29                 IF G^.PNO[G^.LNK[J,1]]=G^.PNO[G^.LNK[J,2]] THEN(* AND THE DUMMY FAMILIES OF THE ARGS ARE THE SAME *)
( 73)   30                 FOR K:=1 TO 2 DO BEGIN
( 73)   31                    M:=1;                 (* CHECK OUT EACH OF THESE LINKS *)
```

```
( 73)   32                  WHILE G^.LNK[G^.LNK[J,K],M]<>0 DO  (* CHECK EACH OF THE LINKS OF THE KTH ARG *)
( 73)   33                  BEGIN
( 73)   34                     L:=G^.LNK[G^.LNK[J,K],M];   (* COMPUTE THE MTH LINK OF THE KTH ARG OF OUR FUNCTION *)
( 73)   35                     IF (G^.PNO[L]=I)AND(J<>L) THEN   (* IF THE LINK IS OUR GUY, BUT WRONG ORDER OR *)
( 73)   36                        IF (L<J)OR(G^.LNK[L,K]<>G^.LNK[J,K]) THEN  (* ORDER LESS THAN OURS *)
( 73)   37                        GOTO 2          (* NO GOOD *)
( 73)   38                        ELSE M:=M+1     (* CONTINUE CHECKING OUT THE LINKS *)
( 73)   39                     ELSE M:=M+1;       (* CONTINUE CHECKING OUT THE LINKS *)
( 73)   40                     END;    (*ADD NODE TO GRAPH*)
( 73)   41
( 73)   42                  L:=1;
( 73)   43                  WHILE G^.LNK[L,1]<>0 DO  (* FIND THE FIRST EMPTY ROW IN THE GRAPH *)
( 73)   44                  L:=L+1;                  (* ?? FUNNY WAY OF DOING IT... *)
( 73)   45                  G^.PNO[L]:=S.NELT-2+K;  (* POINT TO THE FUNCTION NAME *)
( 73)   46                  G^.VBL[L]:=FALSE;     (* ITS NOT A VARIABLE *)
( 73)   47                  G^.ORDIRR[L]:=FALSE;  (* ORDER IS IMPORTANT *)
( 73)   48                  G^.VAL[L]:=[1];       (* ITS TRUE *)
( 73)   49                  G^.LNK[L,1]:=G^.LNK[J,K];  (* POINT IT AT THE RIGHT VARIABLE *)
( 73)   50                  G^.LNK[G^.LNK[J,K],M]:=L;  (* AND PLACE THE RIGHT BACK POINTER *)
( 73)   51  2:
( 73)   52                  END;(*FOR K*)
( 73)   53
( 73)   54               G:=G^.NXTN;
( 73)   55               END;(*WHILE*)
( 73)   56
( 73)   57            END;(*FOR I*)
( 73)   58
( 73)   59         END;(*  ADDML  *)
( 74)    1   *DECK COVER
( 74)    2   PROCEDURE COVER
( 74)    3        (VAR ES          : INTEGER);       (* COVER THE RULES IN EVENT SET ES AGAINST ALL OTHER RULES *)
( 74)    4   LABEL
( 74)    5         1,
( 74)    6         2;
( 74)    7   VAR
( 74)    8         G,Q,P,OPSTAR    : GPTR;
( 74)    9         I,J,K,NUMF1     : INTEGER;        (* RETURNS THE NUMBER OF THE EVENT SET COVERED IN "ES" *)
( 74)   10    (*
( 74)   11                  I N D U C E - 1  --  COVER  --  ABSORB
( 74)   12
( 74)   13    *)
( 74)   14   *CALL ABSOURB
( 74)   15    (*ABSOURB*)
( 74)   16    (*                I N D U C E - 1  --  COVER  --  TOOSMALL
( 74)   17    *)
( 74)   18   *CALL TOOSMALL
( 74)   19
( 74)   20    (*1
( 74)   21                  I N D U C E - 1  --  COVER(MAIN CODE)
( 74)   22    *)
( 74)   23   BEGIN
( 74)   24         IF INFILE=1 THEN READ(CFILE,ES) ELSE READ(IFILE,ES);
( 74)   25         MST.NMST:=0;                     (* SET # OF META SELECTORS TO ZERO *)
( 74)   26         IF MST.METATRIM<>0 THEN          (* IF METATRIM IS NON-ZERO, GENERATE META SELECTORS *)
( 74)   27         ADDMETA;       G:=COVSET;
( 74)   28         IF G<>NIL THEN                   (* IF THERE IS A PREVIOUS COVER SET FROM LAST TIME *)
( 74)   29         BEGIN                            (* RELEASE IT *)
( 74)   30            WHILE G^.NXTN<>NIL DO G:=G^.NXTN;
```

```
( 74)   31          G^.NXTN:=FREEG;       FREEG:=COVSET;
( 74)   32          END;
( 74)   33        COVSET:=NIL;                        (* CLEAR COVSET *)
( 74)   34        NUMF1 := 0;                         (* COUNT NUMBER OF RULES IN F1 *)
( 74)   35        G:=GSET;
( 74)   36        WHILE G<>NIL DO BEGIN               (* PUT ALL OF GSET INTO FP *)
( 74)   37          G^.FP:=TRUE;
( 74)   38          IF ES IN G^.ESET THEN NUMF1 := NUMF1 + 1;(* IF IN F1, COUNT IT *)
( 74)   39          G:=G^.NXTN;
( 74)   40          END;
( 74)   41        G:=GSET;
( 74)   42        WHILE G<>NIL DO BEGIN
( 74)   43          IF G^.FP AND (ES IN G^.ESET) THEN(* SELECT A RULE TO COVER *)
( 74)   44          BEGIN
( 74)   45            FOR I:=1 TO GSIZE DO G^.COUNT[I]:=0;(* SET ALL COUNTS TO ZERO *)
( 74)   46            MQ:=NIL;                        (* NO MQ--MQ WILL BE OUR LIST OF CONSISTENT RULES *)
( 74)   47            PSTAR:=NIL;                     (* NO PARTIAL STAR *)
( 74)   48            STAR:=NIL;                      (* NO STAR *)
( 74)   49            NMQ:=0;(*SET UP INITIAL STAR*)
( 74)   50
( 74)   51            IF 10 IN TRACE THEN BEGIN
( 74)   52              WRITELN(OUTPUT,'NOW COVERING EVENT');       PGRAPH(G,S);       EXPLN(10);
( 74)   53              END;
( 74)   54            FOR I:=1 TO GSIZE DO          (* BUILD A "SEED" STAR FROM EACH UNARY SELECTOR *)
( 74)   55            IF (NOT G^.VBL[I])AND(G^.LNK[I, 1]<>0)AND( G^.LNK[I, 2]=0)
( 74)   56              THEN BEGIN(* NOT A VARIABLE, ONE DUMMY ARGUMENT *)
( 74)   57                NEWG(G1);                   (* MAKE A COPY OF THIS SELECTOR ON STAR *)
( 74)   58                J:=G1^.RNO;                                   G1^:=G^;
( 74)   59                G1^.COUNT[I]:=1;            (* SET COUNT OF THIS SELECTOR TO 1 *)
( 74)   60                G1^.RNO:=J;                                   G1^.NXTN:=STAR;   STAR:=G1;
( 74)   61                FOR K:=1 TO GSIZE DO        (* CLEAR THE REMAINDER OF THE LINKS *)
( 74)   62                FOR J:=1 TO MLNK DO G1^.LNK[K,J]:=0;
( 74)   63                J:=1;
( 74)   64                WHILE G^.LNK[I,J]<>0 DO (* COPY THE LINKS OF THE DESCRIPTOR WE WANT *)
( 74)   65                BEGIN
( 74)   66                  G1^.LNK[I,J]:=G^.LNK[I,J];      G1^.LNK[G1^.LNK[I,J],1]:=I;(* SET BACK POINTERS OF ITS DUMMIES *)
( 74)   67                  G1^.COUNT[G1^.LNK[I,J]]:=1;  (* AND INCLUDE THEM IN THE GRAPH AS WELL *)
( 74)   68                  J:=J+1;
( 74)   69                  END;
( 74)   70                END;                        (* INITIALIZATION IS DONE *)
( 74)   71  2:          G1:=STAR;                     (* MAIN LOOP, LOOP UNTIL WE HAVE NCONSIST RULES *)
( 74)   72              IF 1 IN TRACE THEN BEGIN
( 74)   73                WRITELN(OUTPUT, 'THE FOLLOWING FORMULAS', ' ARE IN THE UNTRIMMED STAR');      EXPLN(1);
( 74)   74                END;
( 74)   75              WHILE G1<>NIL DO              (* COMPUTE COST FUNCTIONS FOR EACH RULE IN UNTRIMMED STAR*)
( 74)   76              BEGIN
( 74)   77                FOR J:=1 TO PRM.NF DO COSTG(G1,PRM.CSTF[J]);
( 74)   78                G1^.FP:=TRUE;               (* ALSO SET FP FLAGS FOR ABSORPTION BELOW *)
( 74)   79                IF 1 IN TRACE THEN PGRAPH(G1,S);
( 74)   80                G1:=G1^.NXTN;
( 74)   81                END;(*ABSOURPTION *)
( 74)   82
( 74)   83              ABSOURB(STAR);                (* ELIMINATE DUPLICATE RULES--SHOULD WE DO THIS *)
( 74)   84                                            (* BEFORE DOING COSTS? *)
( 74)   85              TRIMG(STAR,PRM.MAXSTAR);    (* TRIM STAR TO MAXSTAR BEST RULES *)
( 74)   86              IF 1 IN TRACE THEN BEGIN
( 74)   87                WRITELN(OUTPUT, 'THE FOLLOWING FORMULAS REMAIN AFTER TRIMMING');
( 74)   88                END;
```

```
( 74)   89              IF (NMQ>=PRM.NCONSIST)OR (STAR=NIL) THEN(* IF WE HAVE NCONSIST RULES, OR NOTHING, *)
( 74)   90              GOTO 1;                         (* JUMP OUT OF LOOP *)
( 74)   91              G1:=STAR;                       (* GROW A NEW STAR *)
( 74)   92              WHILE G1<>NIL DO                (* GROW A PARTIAL STAR FOR EACH RULE IN STAR *)
( 74)   93              BEGIN
( 74)   94                 OPSTAR:=PSTAR;               (* SAVE OLD PARTIAL STAR *)
( 74)   95                 NEWGP(PRM.ALTER,G1,G,PSTAR);(* GROW NEW PARTIAL STAR ONTO HEAD OF OLD *)
( 74)   96                 IF PRM.DESCTYPE=CHARACTERISTIC THEN
( 74)   97                    IF TOOSMALL(OPSTAR,PSTAR) THEN ADDTOMQ(G1);
( 74)   98                 IF 1 IN TRACE THEN PGRAPH(G1,S);(*ABSOURPTION *)
( 74)   99
( 74)  100                 P:=PSTAR;                    (* REMOVE LESS GENERAL RULES COVERED BY OTHERS *)
( 74)  101                 WHILE P<>OPSTAR DO           (* FROM THE NEW PARTIAL STAR *)
( 74)  102                 BEGIN                        (* BY SCANNING PSTAR UNTIL WE HIT OPSTAR *)
( 74)  103                    Q:=OPSTAR;
( 74)  104                    WHILE Q<>NIL DO BEGIN
( 74)  105                       IF SUBG1(P,Q,0,AQP.FREEC,TRUE,NIL) THEN (* IF P COVERS Q *)
( 74)  106                       Q^.FP:=FALSE           (* REMOVE Q *)
( 74)  107                       ELSE IF SUBG1(Q,P,0,AQP.FREEC,TRUE,NIL) THEN(* IF Q COVERS P *)
( 74)  108                       P^.FP:=FALSE;          (* REMOVE P *)
( 74)  109                       Q:=Q^.NXTN;
( 74)  110                       END;
( 74)  111                    P:=P^.NXTN;
( 74)  112                    END;
( 74)  113                 G1:=G1^.NXTN;
( 74)  114                 END;(* WHILE G1<>NIL *)
( 74)  115               (*WHILE G1<>NIL*)
( 74)  116
( 74)  117               (* RETURN CURRENT STAR TO FREE LIST*)
( 74)  118
( 74)  119              G1:=STAR;
( 74)  120              WHILE G1^.NXTN<>NIL DO G1:=G1^.NXTN;(* FIND TAIL OF CURRENT STAR *)
( 74)  121              G1^.NXTN:=FREEG;       FREEG:=STAR;       STAR:=PSTAR;       PSTAR:=NIL;
( 74)  122              GOTO 2;                         (* LOOP AGAIN UNTIL NCONSIST RULES OR END OF RULES *)
( 74)  123            (*NOW HAVE MQ LIST OF CONSISTENT FORMULAS-- APPLY AQ PROCEDURE *)
( 74)  124  1:          G1:=MQ;
( 74)  125              IF 2 IN TRACE THEN BEGIN
( 74)  126                 WRITELN(OUTPUT,'THE CONSISTENT FORMULAS:');      EXPLN(2);
( 74)  127                 END;
( 74)  128              WHILE G1<>NIL DO BEGIN
( 74)  129                 IF 2 IN TRACE THEN BEGIN
( 74)  130                    WRITELN(OUTPUT,'BEFORE AQ:');       PGRAPH(G1,S);
( 74)  131                    END;
( 74)  132                 AQSET(GSET,ES,G1);          (* APPLY AQ PROCEDURE, EXTENT G1 AGAINST FO *)
( 74)  133                 FOR I:=1 TO PRM.NF DO       (* RECOMPUTE ALL COSTS EXCEPT -3 *)
( 74)  134                 IF PRM.CSTF[I]<>-3 THEN (* "EFFICIENCY" SINCE COST(3) MUST BE ZERO AFTER AQ *)
( 74)  135                 COSTG(G1,PRM.CSTF[I]);
( 74)  136                 IF 2 IN TRACE THEN BEGIN
( 74)  137                    WRITELN(OUTPUT,'AFTER AQ:');       PGRAPH(G1,S);
( 74)  138                    END;
( 74)  139                 G1:=G1^.NXTN;
( 74)  140                 END;
( 74)  141              ABSOURB(MQ);                    (* REMOVE DUPLICATES THAT MAY HAVE CREPT IN FROM AQ*)
( 74)  142              G1:=MQ;
( 74)  143              IF 9 IN TRACE THEN BEGIN
( 74)  144                 WRITELN(OUTPUT,'THE FOLLOWING ARE ALTERNATIVE CONSISTENT GENERALIZATIONS');      EXPLN(9);
( 74)  145                 END;(* TRACE 9 *)
( 74)  146              WHILE G1<>NIL DO BEGIN
```

```
( 74)  147              IF 9 IN TRACE THEN
( 74)  148                 IF G1^.FP THEN PGRAPH(G1,S);
( 74)  149              G1^.COST[3]:=-100;      G1:=G1^.NXTN;(* WHY SET COST TO -100?????*)
( 74)  150              END;   (* I SUSPECT IT WAS INTENDED TO SPEED UP TRIMG--IT DOESNT WORK *)
( 74)  151           TRIMG(MQ,1);               (* TRIM MQ TO THE BEST RULE *)
( 74)  152           IF 3 IN TRACE THEN BEGIN
( 74)  153              WRITELN(OUTPUT,'THE SELECTED MQ FORMULA IS:');      PGRAPH(MQ,S);      EXPLN(3);
( 74)  154              END;
( 74)  155           MQ^.NXTN:=COVSET;          (* PUT IT ON THE COVSET *)
( 74)  156           COVSET:=MQ;                                           G1:=G;
( 74)  157           WHILE G1<>NIL DO           (* REMOVE FROM F1 ALL RULES COVERED BY THE MQ RULE *)
( 74)  158           BEGIN
( 74)  159              IF (ES IN G1^.ESET)AND(G1^.FP) THEN(* IF IN F1 AND STILL "IN" *)
( 74)  160                 IF SUBG1(MQ,G1,0,AQP.FREEC,TRUE,NIL) THEN(* IF MQ COVERS IT *)
( 74)  161                 G1^.FP:=FALSE;       (* REMOVE IT FROM CONSIDERATION *)
( 74)  162              G1:=G1^.NXTN;
( 74)  163              END;
( 74)  164           END;(*IF G^.FP ETC*)
( 74)  165
( 74)  166        G:=G^.NXTN;
( 74)  167        END;(* LOOP UNTIL F1 IS ALL COVERED OR EXHAUSTED *)
( 74)  168  (*WHILE G<>*)
( 74)  169
( 74)  170     WRITELN(OUTPUT,'THE FOLLOWING FORMULAS COVER SET ',ES);      G:=COVSET;(* PRINT COVSET TO USERS *)
( 74)  171     WHILE G<>NIL DO BEGIN
( 74)  172        WRITELN(OUTPUT,'THIS RULE COVERS',ABS(G^.COST[1]),' NEW RULES');      PGRAPH(G,S);      G:=G^.NXTN;
( 74)  173        END;
( 74)  174     IF MST.NMST<>0 THEN            (* PRINT THE META SELECTOR TABLE *)
( 74)  175     PMETAD;
( 74)  176     END;
( 75)    1  *DECK ABSOURB
( 75)    2  PROCEDURE ABSOURB
( 75)    3       (STAR          : GPTR);      (* REMOVE DUPLICATE RULES FROM THE STAR *)
( 75)    4  LABEL
( 75)    5     99;
( 75)    6  BEGIN
( 75)    7     P:=STAR;                       (* BUBBLE OUR WAY THRU THE STAR *)
( 75)    8     WHILE P<>NIL DO BEGIN
( 75)    9        Q:=P^.NXTN;
( 75)   10        WHILE Q<>NIL DO BEGIN
( 75)   11           IF SUBG1(P,Q,0,AQP.FREEC,TRUE,NIL) THEN(* IF P COVERS Q *)
( 75)   12              IF SUBG1(Q,P,0,AQP.FREEC,TRUE,NIL) THEN(* AND Q COVERS P *)
( 75)   13              BEGIN
( 75)   14                 P^.FP:=FALSE ;       (* REMOVE P FROM FP SET *)
( 75)   15                 GOTO 99;             (* SKIP TO NEXT P *)
( 75)   16                 END;(*IF*)
( 75)   17           Q:=Q^.NXTN;               (* ELSE TRY NEXT Q *)
( 75)   18           END;
( 75)   19  99:      P:=P^.NXTN;               (* TRY NEXT P *)
( 75)   20        END;
( 75)   21     END;
( 76)    1  *DECK TOOSMALL
( 76)    2  FUNCTION TOOSMALL
( 76)    3       (VAR OLDSTAR,NEWSTAR: GPTR): BOOLEAN;(* PURPOSE:  TO TEST TO SEE, DURING A CHARACTERISTIC GENERALIZATION,
( 76)    4     IF THE NEWGP PRODUCED ON THE NEWSTAR FAILS TO COVER "MINCOVER".
( 76)    5     IF SO, IT RETURNS "TRUE".
( 76)    6     NEWSTAR IS THE LIST OF RULES AFTER RUNNING NEWGP.  OLDSTAR POINTS
( 76)    7     INTO THAT LIST AT THE POINT WHERE THERE NEWGP ENDS.
```

```
( 76)    8       ALSO REMOVES AND FREES ANY GRAPH WHICH DOES NOT PASS MINCOVER TEST.
( 76)    9    *)
( 76)   10   VAR
( 76)   11         G,G1,LASTG,FIRSTG: GPTR;
( 76)   12         NUMCOVER        : INTEGER;
( 76)   13   BEGIN
( 76)   14         G:=NEWSTAR;       LASTG := NIL;    (* POINTER TO LAST GOOD GRAPH *)
( 76)   15         FIRSTG := NIL;                     (* POINTER TO FIRST GOOD GRAPH *)
( 76)   16         NUMCOVER := ROUND(PRM.MINCOVER*NUMF1/100.0);(* NUMCOVER IS NUMBER OF RULES THAT MUST BE COVERED *)
( 76)   17         WHILE (G<>OLDSTAR) DO BEGIN
( 76)   18            G1:=G^.NXTN;                    (* REMEMBER NEXT GRAPH IN LIST *)
( 76)   19            COSTG(G,1);                     (* COMPUTE COST FUNCTION 1 : NUMBER OF F1 COVERED *)
( 76)   20            IF G^.COST[1]>=NUMCOVER THEN BEGIN
( 76)   21               LASTG:=G;                    (* REMEMBER LAST GOOD GRAPH *)
( 76)   22               IF FIRSTG=NIL THEN FIRSTG:=G;(* REMEMBER FIRST GOOD GRAPH *)
( 76)   23               END
( 76)   24            ELSE BEGIN
( 76)   25               IF LASTG<>NIL THEN LASTG^.NXTN:=G^.NXTN;
( 76)   26               G^.NXTN:=FREEG;(* PLACE ON HEAD OF FREE LIST *)
( 76)   27               FREEG:=G;
( 76)   28               END;(*IF COST *)
( 76)   29            G:=G1;                          (* POINT TO NEXT GRAPH IN LIST *)
( 76)   30            END;(* WHILE ...*)
( 76)   31         IF FIRSTG=NIL THEN BEGIN
( 76)   32            TOOSMALL:= TRUE;       NEWSTAR:=OLDSTAR;
( 76)   33            END
( 76)   34         ELSE BEGIN
( 76)   35            TOOSMALL:=FALSE;       NEWSTAR:=FIRSTG;
( 76)   36            END;(* IF FIRSTG NIL *)
( 76)   37         END;(* TOOSMALL *)
```

PRINT (V9.1) REQUEST 82/02/25. 09.24.51. /EJECT /OVFL RJE=PPS FORMS = PPS UN=3RSMRSM
FILE: XX /CC

```
******  AQ41088   ***  BIN  88  ******XX(3RSMRSM)          **END**  82/02/25. 13.49.40.  **END**
******  AQ41088   ***  BIN  88  ******XX(3RSMRSM)          **END**  82/02/25. 13.49.40.  **END**
******  AQ41088   ***  BIN  88  ******XX(3RSMRSM)          **END**  82/02/25. 13.49.40.  **END**
******  AQ41088   ***  BIN  88  ******XX(3RSMRSM)          **END**  82/02/25. 13.49.40.  **END**
******  AQ41088   ***  BIN  88  ******XX(3RSMRSM)          **END**  82/02/25. 13.49.40.  **END**
******  AQ41088   ***  BIN  88  ******XX(3RSMRSM)          **END**  82/02/25. 13.49.40.  **END**
******  AQ41088   ***  BIN  88  ******XX(3RSMRSM)          **END**  82/02/25. 13.49.40.  **END**

 5694 LINES. CHARGE:(DCS    ,PS2123 ) TAPE CDCTHB FILE: 2.
```

```
DATE..1982/02/25

JOB...07422 CDCTHB 00002          REEL...CDCTHB   START..1743:42   STOP..1744:25
ACCT..                                    PAGES...000100   STATUS...00008000

END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
END-OF-REPORT
```