The method of insertion has been explained previously and is used in the construction of nonloop descriptors to show the point at which the descriptor for a loop may be inserted. Hence, as each arc is processed, the node to which the arc is incoming is checked against the Loop Base Table to determine if that node is a loop base. If it is, the proper designator (LOOP x) is concatenated onto both the control expression and the kernel expression to signify a point of insertion.

The generator processes all ancestor lists for a given exit node in a similar manner. When all ancestor lists for an exit node have been processed, the generator searches the Assignment Table for other exit nodes to process. When all exit nodes in the table have been processed, the case description is complete.

Output of the Case Description

So far in this chapter, we have seen how we are able to derive output. What is explained here is the exact content of the output.

Output is in the form of basic descriptors. These descriptors are in an abbreviated form, with each

expression consisting of a list of labels representing the statements which would be included in the expression. Hence, the first thing to output for any case description is the table of statements and associated labels which was built in the first pass. This table is used to expand the control and kernel expressions of each descriptor. A sample table is shown in Table 3.2.

TABLE 3.2

Subscript	Expression	Tag	Type
1	ENTRY	0	1
2	I ← 1	3	3
3	SUM ← 0	2	3
4	SUM ← SUM + A(I)	2	3
5	N = I?	1	4
6	I ← I + 1	3	3
7	output ← SUM	2	3
8	EXIT	0	5

The descriptors themselves come in two forms, one for nonlooping paths and one for looping paths. Both forms consist of the control statement list and the kernel statement list. Both are ready for output when they have been created. The descriptors for the looping paths are

output first as they are created first in order to enable easier referencing. Each set of descriptors are labelled to show which loop base they describe. The descriptors for the nonlooping paths are also output as they are created, completing the case description. Examples of these output descriptors may be found in the next chapter.

When this has been completed, the case description is complete. The last set of descriptors, by referencing the loop descriptors, are able to determine any path through the program, and, hence, become the case description for the program.

C H A P T E R I V

EXAMPLES

This chapter takes a sample program and derives its case description using the case description generator. The program is input to the generator in flowchart form and the resulting output is the case description for that program.

Input

The program to be processed is the Wensley Division Algorithm. The flowchart for this program is shown in Figure 4.1. The code representing this flowchart is shown in Figure 4.1a.

The generator accepts this flowchart as input. The generator makes a preliminary pass of this graph and produces the Assignment Table shown in Table 4.1. This table contains the expression for each node in G, the type of expression for each, and its respective tag. The label which is associated with each expression is the subscript corresponding to its entry in the table. Using these

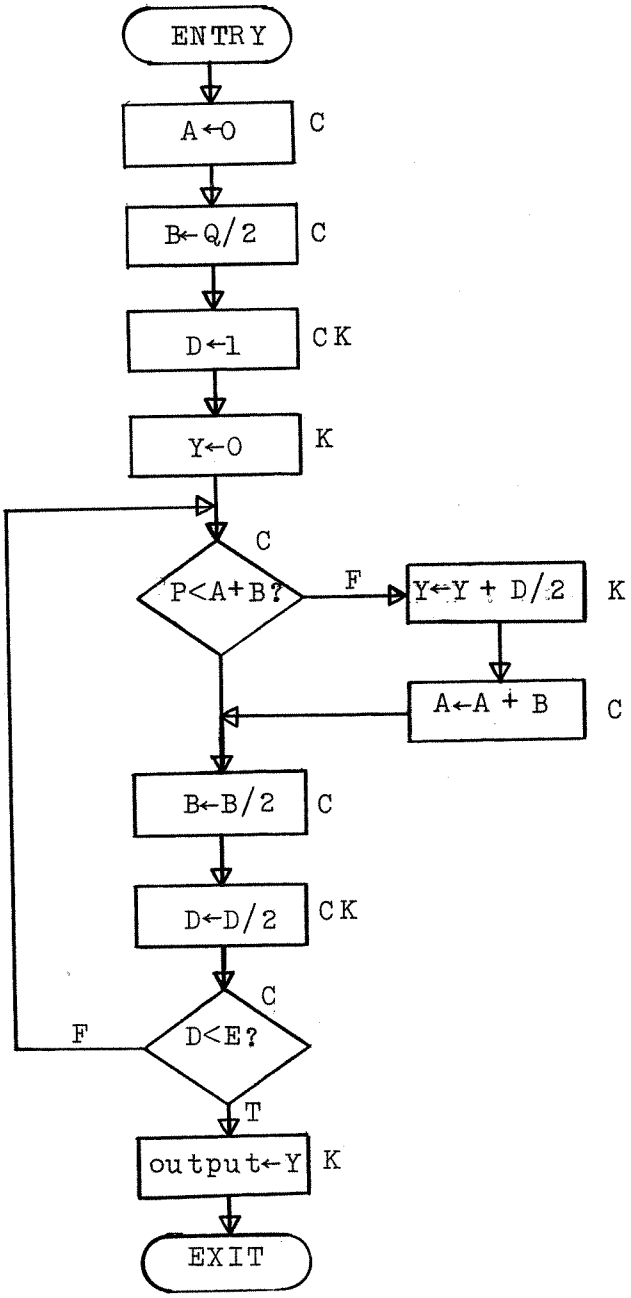
ENTRY
A←0
C
B←Q/2
C
D←1
CK
Y←0
K
C
P<A+B?
F
Y←Y + D/2
K
A←A + B
C
B←B/2
C
D←D/2
CK
C
D<E?
F
T
output←Y
K
EXIT

FIGURE 4.1

TABLE 4.1

Subscript	Expression	Tag	Type
1	ENTRY	A	
2	A←0	F	C
3	B←Q/2	F	C
4	D←1	F	CK
5	Y←0	F	K
6	P<A + B	B	C
7	Y←Y + D/2	F	K
8	A←A + B	F	C
9	B←B/2	F	C
10	D←D/2	F	CK
11	D<E	B	C
12	output←Y	F	K
13	EXIT	Z	

```
      FUNCTION SUB(P, Q, E)
      A = 0
      B= Q/2
      D = 1
      Y = 0
10    IF(P.LT.A+B) 30,20
20    Y = Y + D/2
      A = A + B
30    B = B/2
      D = D/2
      IF(D.LT.E) 40,10
40    SUB = Y
      RETURN
      END
```

Figure 4.1a

labels, G has been changed from its original representation to the one depicted in Figure 4.2.

Initialization

When the Assignment Table and transformation of the input graph is complete, the generator begins the initialization routine. In this routine, the generator sets

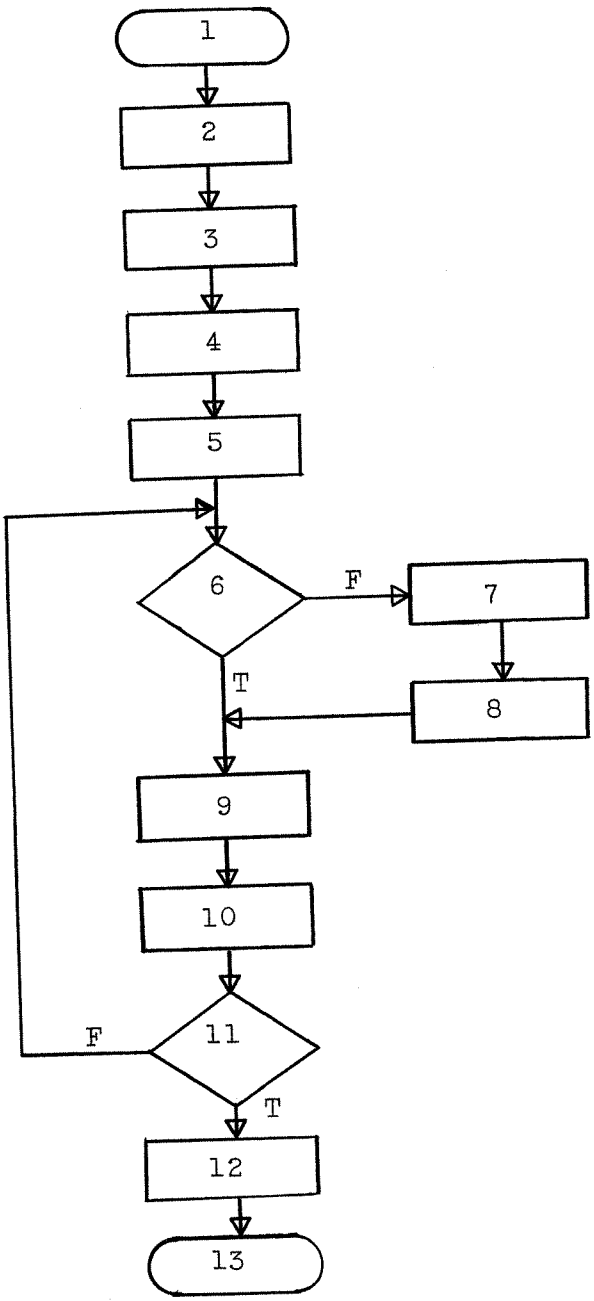

FIGURE 4.2

up the workspace in which the tree is to be built and begins construction of the tree. It creates a node corresponding to the ENTRY node of G and then seeks the next statement for which a node is to be created, i.e., the initial branch statement in the program. To do this, the initialization routine calls a second routine, known as the Direct Descendant algorithm, which finds the direct descendant for the ENTRY node. This routine returns the direct descendant, Q, of the ENTRY node and two lists, the control list, C, containing all control statements between the ENTRY node and Q, and the kernel list, K, containing all kernel statements between the ENTRY node and Q. For the ENTRY node, Q = 6, C = (2, 3, 4), and K = (4,5). Since statement 6 is the first branch statement encountered in any path through the program, the node which is created to represent it becomes the first node representing a branch statement. Its type in the Assignment Table is changed from 'B' to 'I' to denote the initial branch statement of the program. A node in the tree is created corresponding to this statement and an arc is created to attach this node to the original, or ENTRY, node. The two lists, C and K, are attached to the connecting arc as the control and kernel lists for this path. An ancestor list is created for

the initial branch statement and attached to the node in the tree which represents it. The ancestor list, X, consists of the branch list, B, and arch list, A. The branch list contains the label corresponding to the initial branch statement (B = (6)), and the arch list contains the label for the arc connecting the root to this node (A = (201)). Hence, X = ((6)(201)). (The labels for the arcs in this example are completely arbitrary.)

When these actions are complete, the initialization routine is finished. The Assignment Table is complete and is ready to be output for the final case description. The final table is shown in Table 4.2. The tree as it appears after the initialization routine is shown in Figure 4.3.

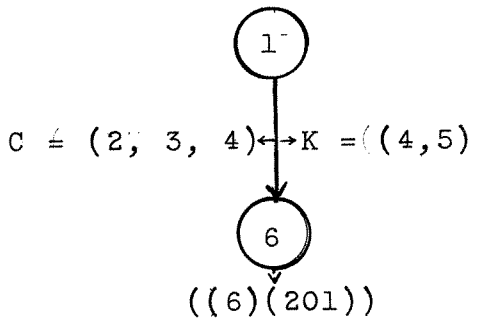

Figure 4.3

TABLE 4.2

Subscript	Expression	Type	Tag
1	ENTRY	A	
2	A←0	F	C
3	B←Q/2	F	C
4	D←1	F	CK
5	Y←0	F	K
6	P<A + B	I	C
7	Y←Y + D/2	F	K
8	A←A + B	F	C
9	B←B/2	F	C
10	D←D/2	F	CK
11	D<E	B	C
12	output←Y	F	K
13	EXIT	Z	

Level 1 Processing

The generator is now ready to find direct descendants of the initial branch statement. The generator calls the Direct Descendant routine, passing to it as parameters the initial branch statement, and the true value for that statement. This routine returns the values Q = 11, C = (6, 9, 10), and K = (10). The node corresponding to the label 11 becomes the direct descendant of node 6. A search of the tree shows that there is no node 11 so a new node must be created. An arc is created between these two nodes and labelled as a true, nonlooping arc. The control list, C = (6, 9, 10), and the kernel list, K = (10), are attached to this arc. The label, 11, is concatenated to a duplicate of the parent nodes's branch list and the arc label, 202, is concatenated to a duplicate of the parent node's arc list. These two lists become the ancestor list, ((6,11) (201, 202)), for the new node. This completes processing for the true branch of the parent node.

The generator calls the Direct Descendant routine, this time passing the false value rather than the true value for the initial branch statement. The values returned by the routine for these parameters are Q = 11, C = (~6, 8, 9, 10), and K = (7, 10). Since node 11 already

exists in the tree, there is no need to create a new node. An arc is created between the parent and its descendant. A search is performed on the ancestor list of the parent using the direct descendant as the search argument. Since no match is found, this arc is classified as a nonlooping arc and is labelled as false, nonlooping. The control list, C = (~6, 8, 9, 10), and the kernel list, K = (7, 10), are attached to this arc. The ancestor list, ((6,11)(201,203)), for the node is determined in the same manner as it was determined for the true branch and attached to the node. Processing for the false branch of the branch statement is complete. Likewise, processing for level 1 is complete. The tree, as it now exists, is shown in Figure 4.4.

Level 2 Processing

The generator now moves to level 2 in the tree. There is only one node, node 11, on level 2 of the tree. The generator begins processing on this node by calling the Direct Descendant routine. The parameters passed are node 11 and the true value. The values returned by the routine for these parameters are Q = 13, C = (11), and K = (12). Since there is no node correspondent to statement 13, a new node is created to represent this statement. A

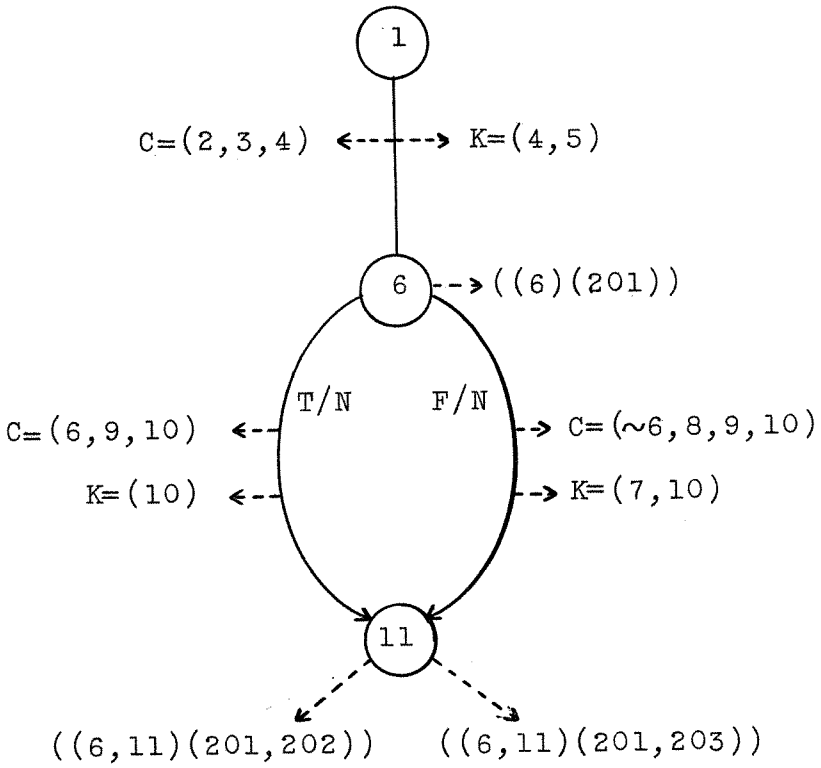
1
C=(2,3,4)
K=(4,5)
6
((6)(201))
T/N
F/N
C=(6,9,10)
K=(10)
C=(~6,8,9,10)
K=(7,10)
11
((6,11)(201,202))
((6,11)(201,203))

FIGURE 4.4

true, nonlooping arc is created to connect this descendant to its parent and the control list, C = (11), and the kernel list, K = (12), are attached to it. Ancestor lists, X and Y, for the new node are created from the ancestor lists of its parents. In this case X = ((6, 11, 13)(201, 202,204)) and Y = ((6, 11, 13)(201,203,204)). Both are attached to node 13 in the tree. Thus, the true branch is complete for node 11.

Now the generator calls the Direct Descendant routine using the false parameter. The values returned are Q = 6, C = (~11), and K = ( ). There is a node 6 in the tree so no new node is created. A search is performed upon the ancestor list of the parent node using the descendant as the argument. This time a match is found so that the arc connecting the two nodes is classified as a looping arc. At the same time, the label 6 is added to the Loop Base Table. An arc connecting the two nodes is created and the appropriate control and kernel lists are attached to it. In this case, the kernel list is empty. The appropriate branch and arc labels are concatenated to duplicates of the parent node's ancestor lists and these new lists are attached to node 6 as ancestor lists. Processing is complete for node 11, and, hence, for level 2 of the tree.

The tree after this step of the processing is shown in Figure 4.5 and the Loop Base Table is shown in Table 4.3.

LOOP BASE TABLE

6

Table 4.3

Completion of the Tree

The generator proceeds to level 3 in the tree. There it finds only one node, and this node is of type Z. No other nodes in the tree need to be processed. Hence, the tree is complete and ready for the derivation of the case description.

Derivation of the Case Description

Upon completion of the creation of the tree, the generator begins derivation of the case description. The Assignment Table is used for reference purposes in the

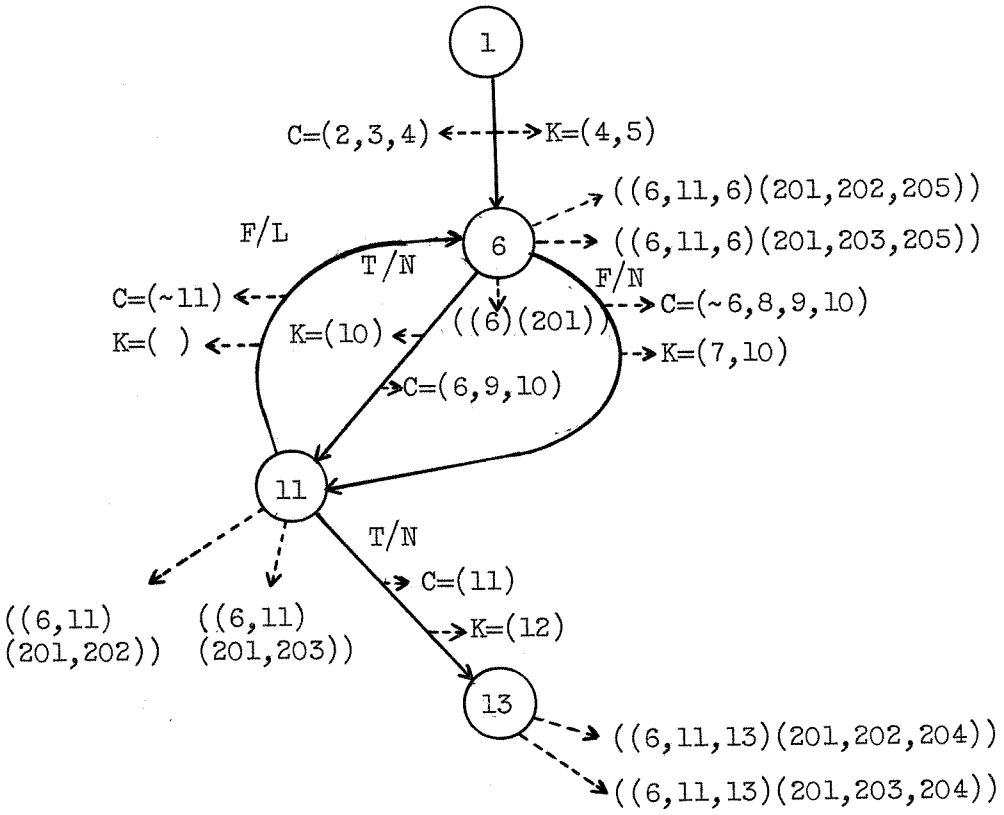
1
C=(2,3,4)
K=(4,5)
((6,11,6)(201,202,205))
6
((6,11,6)(201,203,205))
F/L
T/N
F/N
C=(~11)
K=( )
K=(10)
((6)(201))
C=(~6,8,9,10)
K=(7,10)
C=(6,9,10)
11
T/N
C=(11)
K=(12)
((6,11)
(201,202))
((6,11)
(201,203))
13
((6,11,13)(201,202,204))
((6,11,13)(201,203,204))

FIGURE 4.5

case description. As soon as it is complete, it is output as the beginning of the case description. Its format is shown in Table 4.2

Looping Paths

The generator begins derivation with the looping paths in the program. Searching the Loop Base Table, the generator finds the first loop base to be processed, node 6. The generator finds that this loop base has three ancestor lists to be processed. These ancestor lists are ((6)(201)), ((6, 11, 6)(201, 202,205)), and ((6, 11,6)(201, 203, 205)), which shall be referred to as A, B, and C, respectively.

In processing for loop descriptors, the generator must find ancestor lists which are determined by the paths through the loop. Hence, the generator must search the arc list of each ancestor list for an arc whose from node (i.e., the node from which the arc is outgoing) is equal to the loop base. Since the to node (i.e., the node to which the arc is incoming) of the final arc in the arc list is the loop base, finding an arc whose from node is the loop base shows that the ancester list describes a loop path.

The arc list of list A is searched first. The from node of arc 201 is node 1. Since no other arcs exist in the list, list A does not describe a loop path.

Next the arc list of list B is searched. The from node of arc 201 is 1. Since the test fails, the generator moves to the next arc in the arc list. The from node of arc 202 is node 6. Since node 6 is a member of the Loop Base Table, this means that this list is descriptive of a loop path and the generator sets up the empty control list, C, and the empty kernel list, K, with which it builds the descriptor for the path. To the respective lists, the generator concatenates the tree control and tree kernel lists which are attached to the arc. The generator now moves to the next arc in the arc list, arc 205. From it, the generator retrieves the tree control and three kernel lists, concatenating them to the appropriate lists. The generator now seeks another arc in this path through the loop but finds none. Hence, the descriptor is complete for this path and is ready to be output.

The generator now moves to ancestor list C to process. This ancestor list is found to be descriptive of a path through the loop, and, in a similar manner, the descriptor for this path is constructed and output.

The generator seeks other ancestor lists for node 6 but finds none. The generator seeks other loop bases in the Loop Base Table, but, since node 6 is the only loop base in the program, it finds none. Hence, the descriptors for looping paths are complete. The output for loop descriptors is shown in Figure 4.6.

LOOP 6

CONTROL: ((6, 9, 10) (~11))

KERNEL: ((10) ( ))

CONTROL: ((~6), 8, 9, 10) (~11))

KERNEL: ((7, 10) ( ))

Figure 4.6

Nonlooping Paths

The generator now begins processing on nonlooping paths. To do this, the generator must find the exit nodes in the tree. To find them, it accesses the Assignment Table, searching for a type Z node.

The first node found which is type Z is node 13. Node 13 has two ancestor lists, ((6, 11, 13)(201, 202,204)) and ((6, 11, 13)(201, 203, 204)), to be processed. These shall be referred to as list A and list B, respectively.

Processing begins with the arc list of list A. The generator sets up empty control and kernel lists for the processing of this ancestor list. Onto these lists, the generator concatenates the tree control list and the tree kernel list obtained from arc 201. Before moving to the next arc in the arc list, the generator checks the to node of arc 201 against the Loop Base Table to determine if this is a point of insertion for a loop in the program. The to node of arc 201, node 6, is a member of the Loop Base Table so an identifier for an insertion point of a Loop in the path (i.e., LOOP 6) is concatenated onto both the control list and the kernel list. The generator now moves to the next arc (202) in the arc list. The tree control and tree kernel lists obtained from this arc are concatenated onto the respective list. Since the to node of arc 202, node 11, is not a member of the Loop Base Table, the generator moves to the next arc in the list, arc 204. The tree control and kernel lists are concatenated to the respective lists. The to node, node 13, for this arc is checked against the Loop Base Table. Since it is not a member, the generator moves on to the next arc in the arc list. However, since there are no more arcs in the list, the descriptor for this path is complete and is ready to be output.

The generator now processes the second ancestor list, list B. The processing for this list is similar to that of list A and will not be described in detail. Upon its completion, all ancestor lists for node 13 have been processed.

The generator returns to the Assignment Table to find other type Z nodes. Since there are no other type Z nodes in this program, the set of nonlooping descriptors is complete. These descriptors are shown in Figure 4.7.

```
CONTROL:  ((2, 3, 4) LOOP 6 (6, 9, 10) (11))
KERNEL:   ((4, 5) LOOP 6 (10) (12))

CONTROL:  ((2, 3, 4) LOOP 6 ((~6), 8, 9, 10) (11))
KERNEL:   ((4, 5) LOOP 6 (7, 10) (12))
```

Figure 4.7

The complete case description for the Wensley Division Algorithm is shown in Figure 4.8. This example is brief but includes the important aspects of the derivation of a case description.

FIGURE 4.8

CASE DESCRIPTION FOR WENSLEY DIVISION ALGORITHM

Assignment Table Label	Expression	Tag	Type
1	ENTRY		A
2	A←0	C	F
3	B←Q/2	C	F
4	D←1	CK	F
5	Y←0	K	F
6	P<A + B?	C	I
7	Y←Y + D/2	K	F
8	A←A + B	C	F
9	B←B/2	C	F
10	D←D/2	CK	F
11	D<E?	C	B
12	output←Y	K	F
13	EXIT		Z

LOOP BASE TABLE

6

LOOP DESCRIPTORS

LOOP 6

CONTROL: ((6,9,10) ((~11)))
KERNEL: ((10) ( ))

CONTROL: (((~6),8,9,10) ((~11)))
KERNEL: ((7,10) ( ))

NONLOOP DESCRIPTORS

CONTROL: ((2,3,4) LOOP 6 (6,9,10) (11))
KERNEL: ((4,5) LOOP 6 (10) (12))

CONTROL: ((2,3,4) LOOP 6 ((~6),8,9,10) (11))
KERNEL: ((4,5) LOOP 6 (7,10) (12))

C H A P T E R V

IMPLEMENTATION

This chapter describes the implementation of the case description generator. It describes briefly the language used and the routines needed for implementation.

Language and Machine

The generator was implemented at The University of Texas at Austin on the CDC 6600/6400 in use at the University. The language in which it was written is the FORTRAN-based graph processing language GROPE (1), a GRaph OPerations Extension to FORTRAN IV. The language GROPE is designed to perform operations on atoms, nodes, directed arcs, graphs, graph structures, and lists.

The Program

The following paragraphs describe the implementation of the case description generator. They describe the different routines and how they operate together in order to achieve the goal of producing a case description for a program.

MAIN

This is the main program of the generator which calls the other routines to produce the case description. Its main functions are to call the routines to obtain the flowchart, initialize all counters used in the program, build the tree from the initial branch node to completion, and call the routines which determine and print the descriptors. A more detailed description of MAIN follows.

MAIN first sets up the area to be used by the GROPE system to build the derivation tree. It then sets up all the counters and lists which are used by the generator. MAIN then calls GETGR in order to obtain the flowchart for which the case description is to be built. A call to INIT sets up the Assignment Table and starts construction on the derivation tree. A check is also made to determine whether or not a case description would be meaningless, i.e., whether or not the program in question contains a branch statement. If the case description is meaningless, the generator terminates operation with an appropriate message.

The generator begins construction of the remainder of the derivation tree. It builds a list (NMG) which contains the nodes whose direct descendants have not been

found. Upon return from INIT, this list contains the initial branch node. As the direct descendants for a tree node in this list are found, the node is deleted from the list. As direct descendants are found, they are added to the list if they are not already a member of the list. When no other members of this list exist, the tree is complete.

The direct descendant for a node is determined by a call to DDA. An arc is created to the direct descendant. When this has been done for both the true and the false direct descendant for the node, MAIN processes the next member of NMG.

When the tree is complete, MAIN calls the routines to process looping paths and then the routine to process nonlooping paths. When the latter is finished, the case description for the program is complete.

GETGR

This routine reads in the flowchart for which a case description is to be produced. In this implementation, all data is read from cards. The routine first reads in the number of nodes to be created. The expressions from the flowchart and their tags are each read in and a node

is created in memory to represent them. Each card for a node consists of a ten character field for the expression followed by a one character field for the tag. After all the nodes are read in, the number of arcs to be created is read in and then the arcs to connect the nodes are read in. Each card for an arc consists of a two digit from node field (the node from which the arc is outgoing), a two digit to node field (the node to which the arc in incoming), and a one digit value for the arc (0, 1, or 2 representing none, true, and false). The two digit node labels are determined by the user. For each arc read in, an arc is created in memory connecting the nodes already in memory. When this routine is complete, it has created, in effect, a flowchart in memory.

INIT

This routine performs the initialization procedures for the generator. It classifies each node in the flowchart according to the number of outgoing arcs. It creates the root of the derivation tree and determines its direct descendant. If the direct descendant is an exit node, this routine prints a message stating that the case description is trivial and returns to the main program. If

a meaningful case description can be found, this routine creates a node for the initial branch and creates an arc connecting it to the root. It also adds the initial branch node to the list of tree nodes to be processed by the MAIN program.

DDA

This routine determines the direct descendant for a tree node dependent upon the branch taken from that node. This routine is an implementation of the Direct Descendant algorithm described earlier. The input parameters to this routine are the node for which a direct descendant is to be found and the value assigned to it. The output parameters are the direct descendant, the tree control list and the tree kernel list for the arc connecting these two nodes.

DIRADD

This routine creates the ancestor lists for direct descendants from the ancestor lists of its parent. The routine adds the representation of the direct descendant and the representation of the connecting arc to the

branch list and the arc list of each ancestor list. This new ancestor list is then used as an argument to search the descendant node's ancestor lists in order to prevent duplicate ancestor lists. If it is not a duplicate, it is hung from the node as another ancestor list. This routine also checks the branch list to determine if the node is a loop base and adds the node to the Loop Base Table if this situation occurs.

PROPA

This routine propagates ancestor lists when the direct descendant of a node has already been created in the tree. After the ancestor list for the direct descendant has been hung from that node, this routine adds an appropriate ancestor list to its descendants determined by the path now being processed. When no other descendants exist, the routine is complete. Processing is done much as in MAIN, with a list which is added to and deleted from as new descendants are found and old ones are processed.

NUMOCC

This routine determines the number of occurrences of an element in a list. It is called by PROPA in order to

determine which ancestor lists are not to be propagated from the parent to its direct descendant, i.e., which ancestor lists determine the looping situation in the parent.

LOP1

This routine finds the descriptors for all looping paths in the program being processed. It uses the Loop Base Table to determine the loop bases in the program. When one is found, the routine searches the ancestor lists of the loop base to determine which ancestor lists are the results of looping paths. When one of these lists is found, a descriptor is built for this path and printed in the form of two lists, representing the control expression and the kernel expression for that path. Processing continues for this loop base in the same manner until all ancestor lists for the loop base have been processed. When this occurs, a new loop base is sought in the table. When no other loop bases are left to be processed, the routine is finished.

NONLOP

This routine determines all direct paths through the program. It uses the Assignment Table to find the exit

nodes in the tree. When an exit node is found, its ancestor lists are used to build descriptors. Starting with empty control and kernel expressions, the routine concatenates the tree control and tree kernel lists to form the final descriptor. The tree control and tree kernel lists are obtained from the arcs traversed in a path from the root to this exit node. These arcs are found by processing the arc list from beginning to end. When the end of an arc list is reached, the descriptor is complete and the routine prints it in the form of two lists, representing the control expression and the kernel expression. When all exit nodes have been processed in such a manner, the routine and the case description are complete.

OUT 1

This routine produces all output from the case description generator. It is called by LOP1 and NONLOP to output the control and kernel expressions of the description produced by these two routines. During the initial call of this routine, it prints the Assignment Table and Loop Base Table which have been constructed by the generator.

C H A P T E R V I

CONCLUSION

Future of the Generator

The case description generator described in this paper can become a useful tool for the users of computing machinery. The generator accepts input in flowchart form. To make use of the generator, the user must translate the program he wishes to analyze into a form which is acceptable to the generator. The routine now used reads in individual nodes of the flowchart creating nodes in memory to represent them. It also reads in the arcs connecting these nodes and creates corresponding arcs in memory. The conversion by hand from a flowchart to acceptable input for the generator becomes quite a burdensome task for larger programs. In the future, a routine to create the flowchart in memory is to be integrated into the system in order to ease the task of the user.

The output of the generator, as it now exists, also proves difficult to interpret for larger programs. Each numerical representation of an expression must be looked up in the Assignment Table to determine the correct

control or kernel expression. With smaller programs, i.e., those with short descriptors, the translation is easily done, but as the length of the descriptor grows, the difficulty of the translation increases proportionately. An output routine which performs this translation is also to be integrated into the system in the future, also easing the user's task.

Uses of the Generator

These two routines make the generator an extremely useful tool. Of special interest is its use as an aid in proving the correctness of programs. With extensions, it is able to analyze programs in a more primitive state, i.e., that of a flowchart, and show that the result of that program is with a given set of input and a given control expression. The user is then able to determine from the different descriptors for his program whether or not it meets the specifications desired for that program. In other words, he is able to determine how correct his program is.

Also of interest is the use of the generator in the area of analysis and debugging of programs. Generally, when a program is first written, the author of that program has some idea of what the output for that program is

to be under a given set of conditions. Using the case description generator, he may submit the flowchart (or, if a module is available which is able to translate code into a flowchart, he may submit the coded program) and determine how close the program comes to the desired result. Likewise, at any point in debugging of a partially tested program, he may submit the flowchart (or code) of all or parts of the program to the generator and find out whether or not he is achieving his goals. It is in this area in which the author believes the case description generator will prove to be of the most use.

The generator is not limited to the above. Other uses for the case description, and, hence, for the generator which produces it, have been proposed. The generator may be used in equivalence proofs by showing that two programs which result in the same case description are, in fact, equivalent. Also, it is conceivable that, since a flowchart may be manipulated to form a case description, in the same manner, a case description might be manipulated to form a flowchart from which a program might be derived. A user could formulate the case description for the desired program and submit it to this system to obtain the flowchart (or code). Hence, case descriptions may prove useful in program synthesis.

A P P E N D I X

PROGRAM LISTING

```
      PROGRAM MAIN(INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT)
      COMMON/GRAPE/ARRAY(20000)
      COMMON/KASSTAB/KTABLE1(2,100)
      COMMON/NUMNOD/N
      COMMON/NG/MG
      COMMON/NNG/NMG
      COMMON/EXPTAB/EXP(100)
      COMMON/FSGN/F
      COMMON/NX/MX
      COMMON/KKK/JJJ
      COMMON/TREE/G1
      COMMON/JJ/LL
      COMMON/LTAB/LOOPT(100)
      COMMON/NSW1/NSW1
      COMMON/NSW2/NSW2
      COMMON/NSW3/NSW3
      COMMON/NSW4/NSW4
      INTEGER EQUAL
      INTEGER FIRST, CONCAT
      INTEGER QUOTE
      INTEGER Q1,C1
      INTEGER RD1,CREEDR
C*********************************************************************C
C       THIS IS THE DRIVER PROGRAM.  IT CALLS THE ROUTINES TO         C
C       SET UP THE GRAPH AND INITIATE THE TREE.  IT BUILDS THE        C
C       TREE FROM THE INITIAL BRANCH UNTIL COMPLETION.  UPON          C
C       COMPLETION OF THE TREE, IT CALLS THE ROUTINES TO DETERMINE    C
C       AND PRINT THE DESCRIPTORS.                                    C
C*********************************************************************C
      NSW1=0
      LL=1
      NERSW=0
      BEGIN=SETUP(ARRAY,20000,0.15,5000)
      F = ATOM(1H-,1)
      I1=LIST(QUOTE(1),0)
      L2=LIST(QUOTE(2),0)
      I3=CONCAT(I1,I2)
      MX=L2
      MG=MX
      KG=MG
      NMG=MG
      CALL GETGR
      NERSW=0
      CALL INIT(NERSW)
      IF(NERSW.EQ.1) GO TO 9999
      RD1=CREEDR(NMG)
100   K1=ISATND(RD1)
      IF(K1.EQ.RD1) 5000,101
101   Y=TO(RD1)
      K2=IMAGE(Y)
      IF(KTABLE1(1,K2).EQ.5) GO TO 100
      CALL DDA(K2,1.,C1,K1,Q1)
      K10=Q1+100
      K11=K2+100
      K3=MEMBER(QUOTE(Q1),NMG)
      IF(K3.EQ.QUOTE(Q1)) 107,200
107   J2=LIST(QUOTE(Q1),0)
      JJJ=JJJ+1
      LX=LIST(QUOTE(JJJ),QUOTE(0),QUOTE(1),C1,K1)
      ARRAY(JJJ)=CRARC(ARRAY(K11),LX,ARRAY(K10))
      CALL DIRADD(K11,K10,JJJ)
      CALL PROPA(J2)
      GO TO 203
200   NMG=CONCAT(NMG,LIST(QUOTE(Q1),0))
      LX=LIST(QUOTE(Q1),0)
      ARRAY(K10)=CRTSN(LX,G1)
      JJJ=JJJ+1
      LX=LIST(QUOTE(JJJ),QUOTE(0),QUOTE(1),C1,K1)
      ARRAY(JJJ)=CRARC(ARRAY(K11),LX,ARRAY(K10))
      CALL DIRADD(K11,K10,JJJ)
203   CALL DDA(K2,0.,C1,K1,Q1)
      K10=Q1+100
      K11=K2+100
      K3=MEMBER(QUOTE(Q1),NMG)
      IF(K3.EQ.QUOTE(Q1)) 207,300
207   J2=LIST(QUOTE(Q1),0)
      JJJ=JJJ+1
      LX=LIST(QUOTE(JJJ),QUOTE(0),QUOTE(1),C1,K1)
      ARRAY(JJJ)=CRARC(ARRAY(K11),LX,ARRAY(K10))
      CALL DIRADD(K11,K10,JJJ)
      CALL PROPA(J2)
      GO TO 303
300   NMG=CONCAT(NMG,LIST(QUOTE(Q1),0))
      LX=LIST(QUOTE(Q1),0)
      ARRAY(K10)=CRTSN(LX,G1)
      JJJ=JJJ+1
      LX=LIST(QUOTE(JJJ),QUOTE(0),QUOTE(1),C1,K1)
      ARRAY(JJJ)=CRARC(ARRAY(K11),LX,ARRAY(K10))
      CALL DIRADD(K11,K10,JJJ)
303   GO TO 100
5000  CONTINUE
      CALL LOP1
      CALL NONLOP
9999  CONTINUE
      END
```

```
      SUBROUTINE GETGR
      COMMON/GRAPE/ARRAY(20000)
      COMMON/TREE/G1
      COMMON/KASSTAR/KTABLE1(2,100)
      COMMON/NUMNOD/N
      COMMON/EXPTAR/EXP(100)
      COMMON/NG/NG
      COMMON/ITAB/IX(100)
      COMMON/NNG/LNG
      COMMON/KKK/JJJ
      EXTERNAL TRUE
      INTEGER CONCAT
      INTEGER C,Q1
      INTEGER FIRST,Q
C*****************************************************************C
C       THIS ROUTINE CREATES A FLOWCHART IN MEMORY FROM CARD      C
C       INPUT.  IT READS IN THE NODES OF THE FLOWCHART FIRST      C
C       PLACING THE EXPRESSION FOR EACH AND THE TAG (CONTROL,     C
C       KERNEL, ETC.) FOR EACH INTO TABLES.  IT THEN READS IN     C
C       THE ARCS TO CONNECT THESE NODES.                          C
C*****************************************************************C
21    FORMAT(A10,I1)
      A1=ATOM(4HFLOW,1)
      GR = CREGR(A1)
      READ 10,N
10    FORMAT(I2)
      K=1
      DO 900 I=1,N
      READ 21,EXP(I),KTABLE1(2,I)
      IF(I.NE.1) GO TO 98
30    NG=LIST(QUOTE(I),0)
98    CONTINUE
99    IX(I)=LIST(QUOTE(I),0)
      ARRAY(I)=CRISP(IX(I),GR)
      IF(I.EQ.1) 900,101
101   KX=LIST(QUOTE(I),0)
      IX(I)=LIST(QUOTE(IX),QUOTE(I),0)
      NG=CONCAT(NG,KX)
900   CONTINUE
      READ 10,M
      KKK=N+1
      KKM=N+M
      DO 1090 I=KKK,KKM
      READ 20,IA,IB,IV
20    FORMAT(I2,I2,I1)
      IX(I)=LIST(QUOTE(IV),QUOTE(I))
      ARRAY(I)=CRARC(ARRAY(IA),IX(I),ARRAY(IB))
1090  CONTINUE
      RETURN
      END
```

```
      SUBROUTINE INIT(NSW)
      COMMON/GRAPE/ARRAY(20000)
      COMMON/TREE/G1
      COMMON/KASSTAR/KTABLE1(2,100)
      COMMON/NUMNOD/N
      COMMON/EXPTAR/EXP(100)
      COMMON/NG/NG
      COMMON/ITAB/IX(100)
      COMMON/NNG/LNG
      COMMON/KKK/JJJ
      EXTERNAL TRUE
      INTEGER CONCAT,C,G1,FIRST,Q
C*****************************************************************C
C       THIS ROUTINE PERFORMS INITIALIZATION PROCEDURES.  IT      C
C       CLASSIFIES EACH NODE ACCORDING TO TYPE AND GIVES EACH     C
C       NODE A LABEL WITH WHICH IT MAY BE REFERENCED.  IT ALSO    C
C       CONSTRUCTS THE ROOT OF THE DERIVATION TREE AND THE NODE   C
C       IN THE TREE CORRESPONDING TO THE INITIAL BRANCH IN THE    C
C       PROGRAM AND MAKES THE APPROPRIATE HANGINGS.               C
C*****************************************************************C
      DO 2900 I=1,N
      IF(I.EQ.1) 2001,2100
2001  KTABLE1(1,I) = 1
      GO TO 2900
2100  L = LENGTH(RSETO(ARRAY(I)))
      IF(L.EQ.2) 2130,2140
2130  KTABLE1(1,I) = 2
      GO TO 2900
2140  IF(L.EQ.1) 2150,2160
2150  KTABLE1(1,I) = 3
      GO TO 2900
2160  KTABLE1(1,I) = 5
2900  CONTINUE
3000  G1 = CREGR(ATOM(4HTREE,1))
      ARRAY(101) = CRISP(QUOTE(1),G1)
      I = 1
      CALL DDA(1,0,C,K,C)
      IF(KTABLE1(1,0).EQ.5) 3080,3100
3080  PRINT 3086
3086  FORMAT(22HDESCRIPTOR MEANINGLESS)
      NSW=1
      RETURN
3100  KTABLE1(1,Q) = 4
      Q1=Q+100
      ARRAY(Q1)=CRISP(LIST(QUOTE(Q),0),G1)
      LX=LIST(QLOTE(201),QUOTE(0),QUOTE(2),C,K)
      ARRAY(201)=CRARC(ARRAY(101),LX,ARRAY(Q1))
      X1=OBJECT(ARRAY(Q))
      KX1=LOFT(X1,TRUE)
      LNG=CONCAT(LNG,KX1)
      LX1=LIST(LIST(LIST(QUOTE(Q),0),LIST(QUOTE(201),0),0),0)
      KDS=CONCAT(OBJECT(ARRAY(Q1)),LX1)
      JJJ=201
      RETURN
      END
```

```
      SUBROUTINE DDA(DJ,DV,DC,DK,DQ)
      COMMON/GRAPE/ARRAY(20000)
      COMMON/KASSTAR/KTABLE1(2,100)
      COMMON/NX/MX
      COMMON/FSGN/F
      COMMON/ITAB/IX(100)
      INTEGER CREL
      INTEGER TONODE,OBJECT,FIRST,TO
      INTEGER DJ,DQ,DV,X,V,DC,DK
      INTEGER CONCAT
C*****************************************************************C
C       THIS ROUTINE DETERMINES THE DIRECT DESCENDANT (DQ) OF     C
C       THE NODE (DJ) USING THE PATH DETERMINED BY DV.  IN THE    C
C       PROCESS, IT CREATES THE CONTROL AND KERNEL EXPRESSIONS    C
C       TO DESCRIBE THIS PATH.                                    C
C*****************************************************************C
      I1=LIST(QUOTE(1),0)
      I2=LIST(QUOTE(2),0)
      I3=CONCAT(I1,I2)
      DC=I2
      I4=LIST(QUOTE(2),0)
      I3=CONCAT(I3,I4)
      DK=I4
      I=1
      IF(DJ.EQ.1) 20,30
20    V = FIRST(RSETO(ARRAY(DJ)))
      GO TO 40
30    IF(DV.EQ.1) 31,35
31    V=LAST(RSETO(ARRAY(DJ)))
      LTEMP=LIST(QUOTE(DJ),0)
      DC = CONCAT(DC,LTEMP)
      GO TO 40
35    V=FIRST(RSETO(ARRAY(DJ)))
      LTEMP=LIST(LIST(F,QUOTE(DJ),0),0)
      DC = CONCAT(DC,LTEMP)
40    X =IMAGE(FIRST(OBJECT(TONODE(V))))
      KX=IMAGE(FIRST(OBJECT(TONODE(V))))
      K=KTABLE1(1,X)
      IF(K.EQ.2.OR.K.EQ.4.OR.K.EQ.5) 100,50
50    LTEMP=LIST(QUOTE(KX),0)
      K=KTABLE1(2,X)
      IF(K.EQ.1) 60,51
51    IF(K.EQ.3) 60,70
60    DC = CONCAT(DC,LTEMP)
      LTEMP=LIST(QUOTE(KX),0)
70    IF(K.EQ.2) 80,71
71    IF(K.EQ.3) 80,90
80    DK = CONCAT(DK,LTEMP)
90    KKK=V
      V=FIRST(RSETO(TONODE(KKK)))
      GO TO 40
100   DQ=X
      RETURN
      END
```

```
      SUBROUTINE DIRADD(L,M,JJJ)
      COMMON/GRAPE/ARRAY(20000)
      COMMON/LTAB/LOOPT(100)
      COMMON/JJ/LL
      EXTERNAL OBJECT
      INTEGER OBJECT, HANG, UNHANG, CONCAT, TO, DELETE
      INTEGER QUOTE
      INTEGER FIRST
      INTEGER RDA,CREEDR
      EXTERNAL FIRST
      EXTERNAL LAST
      EXTERNAL TRUE
C*****************************************************************C
C       THIS ROUTINE USES THE ANCESTOR LISTS OF THE NODE L TO     C
C       CREATE NEW ANCESTOR LISTS FOR THE NODE M.  THESE LISTS    C
C       CONSIST OF ALL NODES ENCOUNTERED AND ALL ARCS TRAVERSED   C
C       FOR EACH PATH UP TO M.                                    C
C*****************************************************************C
      MM=M+100
      KX=0
      N=OBJECT(ARRAY(L))
      RDA=CREEDR(N)
      K=TO(RDA)
      L1=OBJECT(ARRAY(M))
1     L2=ISATND(RDA)
      IF(L2.EQ.RDA) 999,100
100   K=TO(RDA)
      LA=NUMOCC(FIRST(K),IMAGE(LAST(FIRST(K))))
      IF(LA.GT.1) 1,101
101   KK1=LOFT(FIRST(K),TRUE)
      J1=MEMBER(QUOTE(MM),KK1)
      IF(J1.EQ.QUOTE(MM)) 200,300
200   KX=0
      DO 206 IB=1,LL
      IF(LOOPT(IB).NE.MM) GO TO 206
      KX=1
206   CONTINUE
      IF(KX.EQ.1) GO TO 300
      LOOPT(LL)=MM
      LL=LL+1
300   KK2=CONCAT(KK1,LIST(QUOTE(MM),0))
      K2=LOFT(LAST(K),TRUE)
      K2=CONCAT(K2,LIST(QUOTE(JJJ),0))
      KK5=LIST(KK2,K2,0)
      K3=MEMBER(KK5,L1)
      IF(K3.EQ.KK5) GO TO 1
      K=LIST(KK5,0)
      L1=CONCAT(OBJECT(ARRAY(M)),K)
      GO TO 1
999   RETURN
      END
```

```
      SUBROUTINE PROPA(LX)
      COMMON/GRAPE/ARRAY(20000)
      INTEGER RD2,CREEDR
      COMMON/AX/MX
      INTEGER RSETO,A2
      INTEGER EQUAL
      INTEGER RD3
      INTEGER QUOTE
      INTEGER CONCAT
C*********************************************************************C
C        THIS ROUTINE PROPAGATES ANCESTOR LISTS THROUGH NODES         C
C        WHICH HAVE ALREADY BEEN CREATED.                             C
C*********************************************************************C
      RD2=CREEDR(LX)
      M1=MX
1     K1=ISATND(RD2)
      IF(K1.EQ.RD2) 999,100
100   A1=TO(RD2)
      KK3=100*IMAGE(A1)
      A2=RSETO(ARRAY(KK3))
      L1=LENGTH(RSETO(ARRAY(KK3)))
      RD3=CREEDR(A2)
      IF(L1.EQ.0) 200,105
105   Q1=TO(RD3)
      Q2=TONODE(Q1)
      KKK=IMAGE(FIRST(OBJECT(Q2)))
      K4=MEMBER(QUOTE(KKK),LX)
      IF(K4.EQ.QUOTE(KKK)) GO TO 106
      LX=CONCAT(LX,LIST(QUOTE(KKK),0))
106   JJJ=IMAGE(FIRST(OBJECT(Q1)))
      KKK=KKK+100
      CALL DIRADD(KK3,KKK,JJJ)
      K6=ISATND(RD3)
      IF(K6.EQ.RD3) 200,107
107   Q1=TO(RD3)
      Q2=TONODE(Q1)
      KKK=IMAGE(FIRST(OBJECT(Q2)))
      K4=MEMBER(QUOTE(KKK),LX)
      IF(K4.EQ.QUOTE(KKK)) GO TO 108
      LX=CONCAT(LX,LIST(QUOTE(KKK),0))
108   JJJ=IMAGE(FIRST(OBJECT(Q1)))
      KKK=KKK+100
      CALL DIRADD(KK3,KKK,JJJ)
200   GO TO 1
999   RETURN
      END
```

```
      FUNCTION NUMOCC(LX,L)
      INTEGER CREEDR,X1
C*********************************************************************C
C        THIS ROUTINE DETERMINES THE NUMBER OF OCCURRENCES OF L IN    C
C        THE LIST LX.  IT IS CALLED BY PROPA TO DETERMINE WHETHER     C
C        OR NOT AN ANCESTOR LIST SHOULD BE PROPAGATED TO              C
C        THE DESCENDANT.                                              C
C*********************************************************************C
      K=0
      A1=CREEDR(LX)
90    K1=ISATND(X1)
      IF(K1.EQ.X1) 999,100
100   A2=TO(X1)
      IF(L.EQ.IMAGE(X2)) 101,90
101   K=K+1
      GO TO 90
999   NUMOCC=K
      RETURN
      END
```

```
      SUBROUTINE LOP1
      COMMON/GRAPE/ARRAY(20000)
      COMMON/TREE/G1
      COMMON/JJ/LL
      COMMON/LTAB/LOOPT(100)
      COMMON/NSW1/NSW1
      COMMON/NSW2/NSW2
      COMMON/NSW3/NSW3
      COMMON/NSW4/NSW4
      EXTERNAL OBJECT
      EXTERNAL TRUE
      INTEGER CREEDR,CONCAT
C*********************************************************************C
C       THIS ROUTINE DETERMINES ALL LOOPING PATHS IN THE PROGRAM.     C
C       IT USES THE ANCESTOR LISTS OF LOOP BASE NODES TO DETERMINE    C
C       IF A LOOPING PATH IS DESCRIBED.  IF ONE IS, THE ROUTINE       C
C       CREATES AND PRINTS ITS DESCRIPTOR.                            C
C*********************************************************************C
      NSW2=0
      NSW3=0
      XXX=ATOM(4HLOOP,1)
      I=1
      LL1=LL-1
1     IF(I.EQ.LL) GO TO 999
100   M=LOOPT(I)
      NSW4=0
      MM=M+100
      KRD1=CREEDR(OBJECT(ARRAY(MM)))
      Y1=TO(KRD1)
101   K1=ISATND(KRD1)
      IF(K1.EQ.KRD1) 105,110
105   I=I+1
      GO TO 1
110   Y1=TO(KRD1)
      JJ1=LAST(Y1)
      KK=LOFT(JJ1,TRUE)
      KRD2=CREEDR(KK)
111   K2=ISATND(KRD2)
      IF(K2.EQ.KRD2) 101,112
112   Y2=TO(KRD2)
      X1=FRNONE(ARRAY(IMAGE(Y2)))
      A1=EQUAL(X1,ARRAY(MM))
      IF(A1.EQ.X1) 150,111
150   I1=LIST(QUOTE(1),0)
      NC=LIST(QUOTE(2),0)
      NK=LIST(QUOTE(3),0)
      I2=CONCAT(I1,NC)
      I2=CONCAT(I1,NK)
151   KRD3=CREEDR(OBJECT(ARRAY(IMAGE(Y2))))
      Y3=TO(KRD3)
      Y3=TO(KRD3)
      Y3=TO(KRD3)
      Y3=TO(KRD3)
      KY2=LIST(Y3,0)
      NC=CONCAT(NC,KY2)
      Y3=TO(KRD3)
      KY2=LIST(Y3,0)
      NK=CONCAT(NK,KY2)
      X1=TONCE(ARRAY(IMAGE(Y2)))
      A1=EQUAL(X1,ARRAY(MM))
      IF(A1.EQ.X1) 200,175
175   DO 180 J=1,LL1
      IF(IMAGE(FIRST(OBJECT(X1))).EQ.LOOPT(J)) 176,180
176   KKK1=LIST(XXX,QUOTE(LOOPT(J)),0)
      KKK2=LOFT(KKK1,TRUE)
      NC=CONCAT(NC,KKK1)
      NK=CONCAT(NK,KKK2)
180   CONTINUE
      K2=ISATND(KRD2)
      IF(K2.EQ.KRD2) GO TO 101
      Y2=TO(KRD2)
      GO TO 151
200   CALL OUT1(NC,NK,M)
      GO TO 101
999   RETURN
      END
```

```
      SUBROUTINE NONLOP
      COMMON/GRAPE/ARRAY(20000)
      COMMON/TREE/G1
      COMMON/JJ/LL
      COMMON/LTAB/LOOPT(100)
      COMMON/KASSTAR/KTABLE1(2,100)
      COMMON/NUMNOD/N
      COMMON/NSW1/NSW1
      COMMON/NSW2/NSW2
      COMMON/NSW3/NSW3
      COMMON/NSW4/NSW4
      EXTERNAL OBJECT
      EXTERNAL TRUE
      INTEGER CREEDR,CONCAT
C*********************************************************************C
C       THIS ROUTINE DETERMINES ALL DIRECT PATHS THROUGH THE          C
C       TREE.  FROM THESE PATHS, DESCRIPTORS ARE BUILT.  THIS         C
C       ROUTINE USES THE ANCESTOR LISTS OF THE EXIT NODES TO          C
C       FIND THE PATHS.                                               C
C*********************************************************************C
      NSW2=1
      NSW3=0
      XXX=ATOM(4HLOOP,1)
      I=1
      LL1=LL-1
1     IF(I.GT.N) GO TO 999
      IF(KTABLE1(1,I).EQ.5) GO TO 100
50    I=I+1
      GO TO 1
100   II=I+100
      KRD1=CREEDR(OBJECT(ARRAY(II)))
      Y1=TO(KRD1)
101   K1=ISATND(KRD1)
      IF(K1.EQ.KRD1) GO TO 50
      Y1=TO(KRD1)
      JJ1=LAST(Y1)
      KK=LOFT(JJ1,TRUE)
      KRD2=CREEDR(KK)
      I1=LIST(QUOTE(1),0)
      NC=LIST(QUOTE(2),0)
      NK=LIST(QUOTE(3),0)
      I2=CONCAT(I1,NC)
      I2=CONCAT(I1,NK)
102   K2=ISATND(KRD2)
      IF(K2.EQ.KRD2) GO TO 500
      Y2=TO(KRD2)
      KRD3=CREEDR(OBJECT(ARRAY(IMAGE(Y2))))
      Y3=TO(KRD3)
      Y3=TO(KRD3)
      Y3=TO(KRD3)
      Y3=TO(KRD3)
      KY2=LAST(Y3,0)
      NC=CONCAT(NC,KY2)
      Y3=TO(KRD3)
      KY2=LAST(Y3,0)
      NK=CONCAT(NK,KY2)
      X1=TONODE(ARRAY(IMAGE(Y2)))
      DO 180 J=1,LL1
      IF(IMAGE(FIRST(OBJECT(X1))).EQ.LOOPT(J)) 176,180
176   KKK1=LIST(XXX,QUOTE(LOOPT(J)),0)
      KKK2=LOFT(KKK1,TRUE)
      NC=CONCAT(NC,KKK1)
      NK=CONCAT(NK,KKK2)
180   CONTINUE
      GO TO 102
500   CALL OUT1(NC,NK,1)
      GO TO 101
999   RETURN
      END
```

```
      SUBROUTINE OUT1(NC,NK,M)
      COMMON/GRAPE/ARRAY(20000)
      COMMON/TREE/G1
      COMMON/LTAB/LOOPT(100)
      COMMON/NSW1/NSW1
      COMMON/NSW2/NSW2
      COMMON/NSW3/NSW3
      COMMON/NSW4/NSW4
      COMMON/JJ/LL
      COMMON/NUMNOD/N
      COMMON/KASSTAB/KTABLE1(2,100)
      COMMON/EXPTAB/EXP(100)
      EXTERNAL OBJECT
C*********************************************************************C
C       THIS ROUTINE PERFORMS ALL OUTPUT FUNCTIONS FOR THE            C
C       GENERATOR.  ON EACH CALL, IT PRINTS THE CONTROL AND           C
C       KERNEL EXPRESSIONS OF A PARTICULAR DESCRIPTOR.  ON THE        C
C       FIRST CALL, IT PRINTS THE ASSIGNMENT TABLE AND LOOP           C
C       BASE TABLE.                                                   C
C*********************************************************************C
      IF(NSW1.EQ.1) GO TO 7000
      NSW1=1
      PRINT 4000
4000  FORMAT(1H1,15X,16HASSIGNMENT TABLE)
      PRINT 4001
4001  FORMAT(//,5X,5HLABEL,6X,10HEXPRESSION,5X,4HTYPE,5X,3HTAG)
      PRINT 4002
4002  FORMAT(1X,1H )
      DO 9000 I=1,N
22    FORMAT(6X,I2,7X,A10,7X,I1,7X,I1)
9000  PRINT 22,I,EXP(I),KTABLE1(1,I),KTABLE1(2,I)
      PRINT 6000
6000  FORMAT(//,5X,5HKEYS:,/,9X,4HTYPE,26X,3HTAG)
      PRINT 6001
6001  FORMAT(6X,7H1=ENTRY,25X,6H0=NONE)
      PRINT 6002
6002  FORMAT(6X,8H2=BRANCH,24X,9H1=CONTROL)
      PRINT 6003
6003  FORMAT(6X,12H3=ASSIGNMENT,20X,8H2=KERNEL)
      PRINT 6004
6004  FORMAT(6X,16H4=INITIAL BRANCH,16X,18H3=CONTROL & KERNEL)
      PRINT 6005
6005  FORMAT(6X,6H5=EXIT)
      PRINT 5005
5005  FORMAT(//,1X,15HLOOP BASE TABLE)
      K16=LL-1
      DO 5001 K=1,K16
5002  FORMAT(9X,I2)
5001  PRINT 5002,LOOPT(K)
7000  IF(NSW2.EQ.1) GO TO 7500
      IF(NSW3.EQ.1) GO TO 7001
      NSW3=1
      PRINT 6100
6100  FORMAT(///,1X,22H***LOOP DESCRIPTORS***)
7001  IF(NSW4.EQ.1) GO TO 7100
      NSW4=1
      PRINT 6101,N
6101  FORMAT(//,1X,7H**LOOP ,I2,2H**)
7100  PRINT 201
201   FORMAT(/,1X,8HCONTROL:)
      P=PRXPFT(NC,OBJECT)
      PRINT 202
202   FORMAT(1X,7HKERNEL:)
      P=PRXPFT(NK,OBJECT)
      RETURN
7500  IF(NSW3.EQ.1) GO TO 7600
      NSW3=1
      PRINT 6200
6200  FORMAT(///,1X,26H***NON-LOOP DESCRIPTORS***)
7600  PRINT 201
      P=PRXPFT(NC,OBJECT)
      PRINT 202
      P=PRXPFT(NK,OBJECT)
      RETURN
      END
```

B I B L I O G R A P H Y

1. Baron, R. J., Friedman, D. P., Shapiro, L. G., Slocum, J., Graph Processing Using Grope/360, The University of Iowa, Dec., 1973.

2. Kaplan, D. M., "Regular expressions and the equivalence of programs," Jour. Comp. and System Sciences, 3, 4, Nov., 1969.

3. Luckham, D. C., Park, M. R., and Paterson, M. S., "On formalized computer programs," Jour. Comp. and System Sciences, 4, 3, June, 1970.

4. Pratt, T. W., Kernel Equivalence of Programs and Proving Kernel Equivalence and Correctness by Test Cases, University of Texas, Jan., 1971.

5. Pratt, T. W., Case Descriptions of Programs: An Informal Introduction, University of Texas, Sept. 1972.

6. Pratt, T. W., Proposal for an executive, program representation and semantic analysis module, University of Texas, Sept., 1973.