PROGRAM TESTING IN LOOP-EXIT SCHEMES

F.M.BOIAN* and M.FRENŢIU*

Received: March, 3, 1993
AMS Subject Classification: 68Q50, 68Q60

REZUMAT. - Testarea schemelor Loop-Exit. În această lucrare se introduce noţiunea de drum complet într-o schemă Loop-Exit şi se arată importanţa drumurilor complete într-o schemă program pentru testarea programelor. De asemenea, se construieşte un limbaj care generează mulţimea drumurilor complete.

1. Introduction. In this paper we consider the Loop-Exit Schemes as they were defined in [2]. Nevertheless, we impose a minor condition: there is an initial assignement a_0 just at the begining (after START block in the corresponding flowchart), and a final assignment a_f at the end (in front of the STOP block). A and T are the sets of assignment and test symbols, respectively, and $M = A \cup T$. Also, we denote by $SW(S)$ the skeleton word associated to S, and we denote by $D(x\alpha y)$ the direct word from x to y (as in [4]).

To each Loop-Exit Scheme S a language $L(S)$ may be associated. More exactly, we have the following definition:

DEFINITION 1. *The language $L(S)$ associated to the Loop-Exit Scheme S is generated by the following context free grammar* [1,2,3]:

$$G(S) = (\mathrm{N}, \Sigma, \mathcal{P}, \nabla)$$

where

$$\mathrm{N} = \{\nabla\} \cup \{I_j | j>0\} \cup \{L_k | k>0\},$$

$$\Sigma = M \cup \{+,-\}$$

I_j is a nonterminal for IF_j, and L_k and B_k are two nonterminals

* *University of Cluj-Napoca, Departament of Computer Science, 3400-Cluj-Napoca, Romania*

for $LOOP_k$ from the definition of the Loop-Exit Scheme S, ∇ is a new symbol - the axiom of $G(S)$, and the set $\mathcal{P}$ of the productions is constructed by the following rules:

a) ∇ ---> SW(S)

b) the following productions

b1) I_j ---> $b-$

b2) I_j ---> b+SW(α) only if α has not the form $\alpha' EXIT_k$

are in $\mathcal{P}$ if

IF_j b $THEN_j$ α $ENDIF_j$;

is in S.

c) the productions

c1) I_j ---> b+SW(α) if $\alpha \neq \alpha' EXIT_k$;

c2) I_j ---> b-SW(β) if $\beta \neq \beta' EXIT_k$;

are in $\mathcal{P}$ if

IF_j b $THEN_j$ α $ELSE_j$ β $ENDIF_j$;

is in S.

d) if

$LOOP_k$ $\alpha_1\alpha_2\delta$ $ENDLOOP_k$;

is in S then the productions

d1) L_k ---> SW($\alpha_1\alpha_2\delta$)L_k

d2) B_k ---> SW($\alpha_1\alpha_2\delta$)B_k | ϵ

d3) L_k ---> D($LOOP_k$ α_1 IF_j) b+SW(β)

if

α_2 = IF_j b $THEN_j$ β $EXIT_k$; $ENDIF_j$;

or

α_2 = IF_j b $THEN_j$ β $EXIT_k$; $ELSE_j$ γ $ENDIF_j$;

d4) L_k ---> D($LOOP_k$ α_1 IF_j) b-SW(β)

if

α_2 = IF$_j$ b THEN$_j$ γ ELSE$_j$ β EXIT$_k$; ENDIF$_j$;

are in $\mathcal{P}$.

Intuitively, $L(S)$ contains the set of all sequences which can be met during the execution of the scheme.

2. The complete paths in a Loop-Exit Scheme. An important problem in software development is program testing. Testing may be done starting from the specification of the resolved problem, or starting from the text of the program. In the second alternative it is important to know all the paths from the START block to the STOP block of the corresponding flow chart. For this purpose we introduce the notion of complete path in a Loop-Exit Scheme.

DEFINITION 2. *A word* $z = a_{i_1}X_{i_1}a_{i_2}X_{i_2}\ldots a_{i_s}X_{i_s}$ *is a section for S if and only if there is* $w \in L(S)$ *such that:*

a) $w = xzy$

b) $i_j < i_{j+1}$ *for* $j=1,2,\ldots,s-1$

c) *if* $x \neq \epsilon$ *then* $x = x'a_{i_0}X_{i_0}$ *with* $i_0 > i_1$

d) *if* $y \neq \epsilon$ *then* $y = a_{i_{s+1}}X_{i_{s+1}}y'$ *with* $i_s > i_{s+1}$.

The set of all sections is denoted by SEC(S).

The following theorem is proved in [2]:

THEOREM 1. *For each S we have* $L(S) \subset (\mathrm{SEC}(S))$.

DEFINITION 3. *A word* $z \in \mathrm{SEC}(S)$ *is a branch for S if and only if there is* $w \in L(S)$ *such that* $w=zy$. *The set of all branches of S is denoted by* BRA(S).

Next, an algorithm to construct the set BRA(S) is given.

Algorithm 1. Which constructs the set BRA(S), has the following steps:

Step 1. The grammar G_1 has the productions obtained from the productions of $G(S)$ by replacing the productions

$$B_k \dashrightarrow \alpha B_k \mid \epsilon,$$

with the production B_k --> α and in all the other productions which have not this form the metasymbol B_k is replaced by ϵ.

Step 2. Putting off the inaccesible and unseful metasymbols of G_1 we obtained the grammar G_2 [1];

Step 3. The grammar G_3 is obtained from the grammar G_2 by replacing the productions of the form

$$L_k \dashrightarrow \alpha\ L_k$$

by the productions

$$L_k^a \dashrightarrow \alpha$$

where L_k^a is a new metasymbol associated to L_k ;

Step 4. The grammar G_4 is constructed from the grammar G_3 by adding to the productions of G_3 some new productions. If L_k is a recursive symbol in G_2 and A --> $\alpha\ L_k\ \beta$ is in G_3 then add the production $A \dashrightarrow L_k^a$ to G_4. Here L_k^a is the symbol associated to L_k.

Step 5. One computes BRA(S) = $L(G_4)$.

To each metasymbol A of a grammar

$$G=(\mathrm{N},\Sigma,\mathcal{P},\nabla)$$

one can associate the grammar

$$G_A = (\mathrm{N},\Sigma,\mathcal{P},A)$$

which has the metasymbol A as the axiom.

If BRA(A) is the result of the application of the algorithm 1 to the grammar G_A then the following theorem holds [1].

THEOREM 3. *If S is a Loop-Exit Scheme then*

$$\mathrm{SEC}(S) = \mathrm{BRA}(S) \cup \{\mathrm{BRA}(A) \mid A \text{ is recursive in } G_S^r\},$$

where G_S^r is the reduced grammar of the scheme S.

DEFINITION 4. *For each $xy^nz \in L(S)$ with $n \geq 0$ and $y \in \mathrm{SEC}(S)$ the words $w_1=xz$ and $w_2=xyz$ with $x=a_0x_1$ and $z=z_1a_f$ (i.e. which contains the assignments a_0 and a_f) are called complete paths of the Loop-Exit Scheme. The set of all complete paths of S is denoted by $CP(S)$.*

THEOREM 4. *Let G_p be the grammar obtained from G in the following way: if A is a recursive symbol in G and*

$$A \dashrightarrow \alpha A \mid \beta_1 \mid \beta_2 \ldots \mid \beta_k$$

are all the A-productions of G then the A-productions of G_p are

$$A \longrightarrow \beta_1 \mid \beta_2 \ldots \mid \beta_k \mid \alpha\beta_1 \mid \alpha\beta_2 \ldots \mid \alpha\beta_k$$

The language generated by the grammar G_p generates CP(S).

The proof of this theorem follows imediatelly from the definition 4.

To ilustrate these we consider the following Loop-Exit Scheme:

a_1 a_2

LOOP_1

IF_1 a_3 THEN_1 EXIT_1 ENDIF_1

IF_2 a_4 THEN_2 a_5

ELSE_2 a_6 a_7 a_8 ENDIF_2

$\mathrm{ENDLOOP}_1$

a_9

The grammar $G(S)$ and the reduced grammar G_S^r are

$G(S)$	G_S^r
$\nabla \longrightarrow a_1\ a_2\ L_1\ a_9$	$\nabla \longrightarrow a_1\ a_2\ L_1\ a_9$
$L_1 \longrightarrow I_1\ I_2\ L_1 \mid a_3+$	$L_1 \longrightarrow I_1\ I_2\ L_1 \mid a_3+$
$B_1 \longrightarrow I_1\ I_2\ B_1 \mid \epsilon$	$I_1 \longrightarrow a_3-$
$I_1 \longrightarrow a_3-$	$I_2 \longrightarrow a_4+a_5 \mid a_4-a_6a_7a_8$
$I_2 \longrightarrow a_4+a_5 \mid a_4-a_6a_7a_8$	

For this Loop-Exit Scheme we have

$$\mathrm{BRA}(S) = \{\ a_1a_2a_3+a_9\ ,\ a_1a_2a_3-a_4+a_5\ ,\ a_1a_2a_3-a_4-a_6a_7a_8\ \}$$

and

$$SEC(S) = BRA(S) \sqcup \{ a_3{+}a_9 , a_3{-}a_4{+}a_5 , a_3{-}a_4{-}a_6a_7a_8 \}$$

The grammar G_p has the following productions:

v ---> $a_1\ a_2\ L_1\ a_9$

L_1---> $I_1\ I_2\ a_3{+}$ | $a_3{+}$

I_1---> $a_3{-}$

I_2---> $a_4{+}a_5$ | $a_4{-}a_6a_7a_8$

and the set of the complete paths is

$$CP(S) = \{ a_1a_2a_3{+}a_9 , a_1a_2a_3{-}a_4{+}a_5a_3{+}a_9 , a_1a_2a_3{-}a_4{-}a_6a_7a_8a_3{+}a_9 \}.$$

3. Testing a Loop-Exit Program Scheme. Similarly to [6] any Loop-Exit Scheme becomes a Program Scheme if the assignments and test symbols are defined as follows.

Let

$$V = \{v_1, v_2, \ldots, v_m\} = I \sqcup W \sqcup O$$

be a set of variables, where I is the set of input variables, W is the set of working variables, and O is the set of the output variables. We may suppose, as in [9], that the set I, W and O are mutually disjoint. Let

$$F = \{f_1, f_2, \ldots, f_n\}$$

be a set of functional symbols. We suppose that each assignment $a \in A$ is of the form

$$v := f(y_1, y_2, \ldots, y_k)$$

where $f \in F$, $k \geq 0$, $y_1, y_2, \ldots, y_k \in I \sqcup W$, and $v \in W \sqcup O$.

Further, let

$$T = \{t_1, t_2, \ldots, t_r\}$$

be a set of test symbols. We suppose that each test symbol of the

Loop-Exit Scheme is of the form

$$t(y_1, y_2, \ldots, y_k)$$

where $t \in T$, $k \geq 0$, and $y_1, y_2, \ldots, y_k \in I \sqcup W$.

DEFINITION 5. *A Loop-Exit Scheme S is a Loop-Exit program Scheme if the symbols* $a \in M$ *are defined as above, and for any* $w \in L(S)$ *and any* $v \in W$ *if* $w = w_1 aXw_2$ *and a is of the form* $t(\ldots, v, \ldots)$ *or* $u := f(\ldots, v, \ldots)$ *then there is* $a' \in A$ *of the form* $v := f(y_1, y_2, \ldots, y_k)$ *such that* $w = w'a'w''aXw_2$.

As an example, from the Loop-Exit Scheme given above we obtain the following Program Scheme:

```
d:=n1;  i:=n2;
LOOP1
     IF1 d=1 THEN1 EXIT1 ENDIF1
     IF2 d>i THEN2 d:=d-i
            ELSE2 t:=i; i:=d; d:=t ENDIF2
ENDLOOP1
div:=d
```

In other words, the definition 5 asks that any working variable is first initialized and then this variable may be used in computation.

The condition of the definition 5, taken from [6], is very strong. An example of a Loop-Exit Program Scheme which do not satisfy this condition but all variables receive their values before their use, is given in [4]. Also, in [4] is shown that a scheme S is a Program scheme if and only if this condition holds for any $z \in$BRA(S). It follows that if a variable does not satisfy this condition for every $z \in$BRA(S) then it is certainly an uninitialised variable. This fact is very important for the verification of the program corectness. Also, it is important for the programmer to be informed about all the uninitialised variables on some branches of the program.

Testing a program [7] means to observe the results obtained if the program is run for some testing data. A run is needed for each complete path. Therefore, for program testing it is very important to know all of its complete paths.

Knowing a complete path is also useful for choosing the coresponding testing data. If the input variables receives these data the program follows this path. That is, all test conditions met in this path are satisfied.

R E F E R E N C E S

1. Aho A.V., Ullman J.D., *The theory of Parsing, Translation and Compiling*, Prentice Hall Inc., 1972-1973.
2. Boian F.M., *Sisteme conversaţionale pentru instruire în programare*, Teză de doctorat, Cluj-Napoca, 1986.
3. Boian F.M., *Loop-Exit Schemes and Grammars: Properties Flowchartablies*, Studia Universitatis "Babeş-Bolyai", Math.(1986), n0.3, pp. 52-57.
4. Boian F.M., M.Frenţiu, and Z.Kasa, *Parallel execution in Loop-Exit Schemes*, Seminar on Computer Science, Preprint no.9, 1988, pp.3-16.
5. Floyd,R.W. (1967), *Assigning meanings to programs*, in Proc. Symposium App.Math., XIX,(J.T.Schwartz ed.), Providence, Am.Math.Soc.
6. Greibach S., *Theory of Program Structures: Schemes, Semantics, Verification* (Lecture Notes in Computer Science), Springer-Verlag, 1975.
7. J.C.King, *Symbolic Execution and Program Testing*, Comm. ACM, 19 (1976), 7, 385-394.
8. S.Katz, Z.Manna, *Logical Analysis of Programs*, CACM 19(1976), 4, 188-206.
9. Manna, Z. (1974) *Mathematical Theory of Computation*, New York: McGrawHill.