STUDIA UNIV BABEŞ-BOLYAI, MATHEMATICA, XXXVIII, 3, 1993

GENERATING CONTROL STRUCTURES

V. CIOBAN, S. MOTOGNA, V. PREJMEREAN*

Dedicated to Professor Emil Muntean on his 60th anniversary

Received November 18, 1992
AMS subject classification 68N20, 68Q52

REZUMAT. - Generarea structurilor de control. Lucrarea prezintă o modalitate de a defini specificaţiile formale cu ajutorul unei gramatici necontextuale

1. Introduction. The aparition of the programming environments generates an accentuated grow of programmers productivity With such a software instrument many actions can be performed editing a source file, compiling and linkediting of a program, execution, debugging even others facilities for files viewed as entities In fact, the aparition of microcomputers and programming environments made a combination of the programming work with the operating work in a calculus system The abandon of the "batch" working style and working interactively impose a specific training in operating a computer If the first programming environment have had restricted functions, the recent ones, as TURBO PASCAL or BORLAND C (considered in top of the classification), are very complex and are few specialists who can handle them completely However, the programming languages from these environments (PASCAL, C, C++) may be considered universal languages (solve a great number of problems technical, scientifical problems, problems which had to work with many informations and so, with files, graphical problems, object-oriented programming) and, that's

* *"Babeş-Bolyai" University, Faculty of Mathematics and Computer Science, 3400 Cluj-Napoca, Romania*

why handling all of the language facilities became difficult From another point of view languages as PASCAL, C++, COBOL or DBASE IV have thicker instructions, from the syntactical aspect, as FORTRAN We though that an instrument for automatic generation of control structures in a fixed language may be added as an important function in a programming environment

The problem of automatic generation of programs is not recent, and program generators exist in some systems and software products As an example we mention DBASE IV system which has a program generator based on graphical specification

We propose a model for generating some control structures of a program using context free grammars (1) A problem which hasn't been solved efficently is the specification of the structures

2. Control structures. For Dijkstra structures (see for example (2))and for other structures we will introduce the following operators

a) $C(s_1,s_2)$ - operator for concatenation structures s_1 and s_2 in this order ,

b) $\Delta(b,s_1,s_2)$ - operator associated to the complete alternative structure (complete IF) with the semnification

```
IF b THEN
   s1
  ELSE
   s2
ENDIF,
```

c) $\llcorner(b,s)$ - operator associated to the alternative structure with one alternative (simple IF) with semnification IF b THEN s ENDIF,

d) $*(b,s_1, \ ,s_n)$ - operator associated to the generalized alternative structure (CASE)

e) Ʊ(b,s) - operator associated to pretested loop with the semnification

```
WHILE b DO
    s
ENDWHILE,
```

f) Ω(s,b) - operator associated to posttested loop with the semnification

```
REPEAT
    s
UNTIL b.
```

Are required some explanations

- the three Dijkstra are D={ C, Δ, Ʊ} and are considered fundamental, with them any algorithm can be described,
- we asociate operators for structures D'={ C, Δ, ʟ, *, Ʊ, Ω} which are in fact the structures from the PASCAL language,
- any other structure to which a similar operator can be asociated may be simulated with D or D' (for example LOOP-EXIT or LOOP-EXITIF-ENDLOOP structures),
- we may introduce the λ symbol for the empty structure

3. Proprieties of the asociated operators

1 $C(s_1,s_2) \neq C(s_2,s_1)$ - concatenation of structures s_1 and s_2 isn't comutative

2 $C(s_1,C(s_2,s_3)) = C(C(s_1,s_2),s_3)$ - concatenation is asociative

3 $C(s,\lambda) = C(\lambda,s) = s$ - the symbol of the empty structure is playing the role of the neutral element for concatenation

4 $C(\Delta(b,s_1,s_2),s_3) = \Delta(b,C(s_1,s_3),C(s_2,s_3))$ - concatenation is right distributed to alternative

structure

5 $C(s_1,\Delta(b,s_2,s_3)) = \Delta(b,C(s_1,s_2),C(s_1,s_3))$ - concatenation is distributed to left to alternative structure if and only if s_1 structure doesn't have any effect on b predicat

6 $\text{Ʊ}(b,s) = \Delta(b,C(b,\text{Ʊ}(b,s)),\lambda) = \Delta(b,C(s,\Delta(b,C(s,\text{Ʊ}(b,s)),\lambda)),\lambda) =$ - this propriety shows that the three D structure can be reduced to only two structures concatenation and the alternative structure

7 Reducing D' structures to D structures

a) $\text{Ⱡ}(b,s) = \Delta(b,s,\lambda)$

b) $\text{Ⱡ}(b,s) = \Delta(b,s,\text{Ʊ}(c,s))$

c) $*(b,s_1, \; s_n) = \Delta(b_1,s_1,\Delta(b_2,s_2,\Delta(\;\; ,\Delta(b_{n-1},s_{n-1},s_n) \;\;)$

where b is formed from $b_1, \;\; ,b_{n-1}$

d) $\Omega(s,b) = C(s,\text{Ʊ}(\neg b,s))$, where $\neg b$ is the negation of b

8. Some equivalence proprieties

a) $\Delta(b,s_1,s_2) = C(b_1='T',C(\text{Ʊ}(b\wedge b_1,C(b_1='F',s_1)), \text{Ʊ}(b\wedge\neg b_1,C'(b_1='F',s_2))))$

Δ could be reduced to the operators C by introducing a new boolean variable b_1 ('T' is the value TRUE and 'F' is the value FALSE)

b) $\Delta(b,s_1,s_2) = C(\text{Ⱡ}(b,s_1),\text{Ⱡ}(\neg b,s_2))$ mentioning that s_1 doesn't modify b

4. Generating grammars for control structures. With the introduced notation we try to define a grammar which generates programms containing only control structures whose associated operators have been described One may give more than one grammar but we'll reffer only to the structures C, Δ, Ⱡ, Ʊ and Ω

Having n structures s_1, ,s_n (which may be considered the simplest ones, namely attributing) and 2k predicates b_1, b_k and $\neg b_1$, ,$\neg b_k$ we give a grammar which generates all programms over the objects considered above

Let $G = (N,\Sigma,P,S)$, where

$N = \{S,B\}$ is the neterminals set

$\Sigma = \{C\ \Delta\ \text{Ⱡ}\ \mho\ \Omega\ (\ ,\ )\ s_1\quad s_n\ b_1\quad b_k\ \neg b_1\quad \neg b_k\ \}$

is the alphabet of the grammar

P $S \to C(S,S)|\mho(B,S)|\Omega(S,B)|\text{Ⱡ}(B,S)|\Delta(B,S,S)|s_1|\quad |s_n$

$B \to b_1|\quad |b_k|\neg b_1|\quad |\neg b_k$

is the set of production rules

S - is the source symbol, $S \in N$

We consider the following examples

Example 1 The word

$C(s_1,C(s_2,C(\text{Ⱡ}(b_1,s_3),C(s_2,\mho(\neg b_2,s_4)))))$

which belongs to L(G) over $s_1,s_2,s_3,s_4,b_1,b_2,\neg b_1,\neg b_2$ may be obtained through "=>" in this way

$S \Rightarrow C(S,S) \Rightarrow C(S,C(S,S)) \Rightarrow C(S,C(S,C(S,S))) \Rightarrow$

$C(S,C(S,C(\text{Ⱡ}(B,S),C(S,S)))) \Rightarrow C(s_1,C(s_2,C(\text{Ⱡ}(b_1,s_3),C(s_2,\ \mho(\neg b_2,s_4)))))$

and it is equivalent with the following program

```
s1,
s2,
IF b1 THEN s3,
s2,
WHILE ¬b2 DO
      s4
ENDWHILE,
```

Example 2 Let's consider the following word ·

$$C(s_1, \Delta(b_1, (b_2,s_2), \Omega(s_3,\neg b_2)))$$

$\in L(G)$, which is obtained in this way

$$S \Rightarrow C(S,S) \Rightarrow C(S,\Delta(B,S,S)) \Rightarrow C(S,\Delta(B, (B,S),\Omega(S,B))) \Rightarrow$$
$$\Rightarrow C(s_1,\Delta(b_1, (b_2,s_2),\Omega(s_3,\neg b_2)))$$

and it is equivalent to the following program

s_1,
IF b_1 THEN
WHILE b_2 DO
s_2
ENDWHILE
ELSE
REPEAT s_3
UNTIL $\neg b_2$
ENDIF

The introduced grammar has the following properties ,

- is a simple precedence grammar

- there are no conflicts in grammar

We may prove that for any program (written in any language) only with structures C, Δ, ⊾, Ʊ and Ω exists one single word from L(G), which reproduces the program through operators

Different generators may be construct now having as input a word from L(G) and as output a program written in PASCAL, C, C++, COBOL, FORTRAN and so on The problem which hasn't been solved properly is the specification of the word from L(G) at input

R E F E R E N C E S

1 Aho,A V and Ullman,J D - The Theory of Parsing, Translation and Compiling, vol 1 and 2 Englewood Cliffs, New Jersey, Prentice Hall,1972

2 Dahl,O J , Dijkstra E W, Hoare,C A R - Structured programming Academic Press, London, 1972