A METHOD OF ANALYZING PROGRAMS BASED ON A MACHINE LANGUAGE

E. M. Lavrishcheva and E. L. Yushchenko

UDC 681.3.06:51

The apparatus of a machine grammar was proposed in [1] to describe programming languages and solve the problem of syntax analysis and control for a very wide class of languages which are important in practice. In this paper the apparatus is developed and generalized in the direction of using it to solve the problem of the analysis and synthesis in translation. To this end a relation is established between the symbols of the input sequence and their semantic equivalents directly in terms of the generalized rules of a machine grammar. The description of the language with the aid of these rules is a description of a translator for the language into some other language. The translator consists of a sequence of calls of semantic subprograms of which the actual parameters are the symbols of the input sequence.

The account is illustrated by the example of the specification of a generalized machine grammar; the characteristics are given of a typical programming system which is implemented on the basis of the apparatus for the Dnepr-2 computer.

1. Let Σ and Δ be terminal dictionaries of the input and output languages: $W_s = W_S(\nu)$, $s = \{0, 1, 2\}$, $\nu = \{\emptyset, \nu_1, \ldots, \nu_k\}$, $\emptyset \in \Delta$ ($\emptyset$ is the empty symbol).

The description of the translator T in a machine language is specified by the enumeration of a sequence of rules (operators) having one of the following forms:

$$A \overset{W_s(\nu)}{\rightarrow} \delta * \rho * n * M, \tag{1}$$

$$\text{STACK } \beta, \tag{2}$$

$$\text{SE } \alpha. \tag{3}$$

Here A is a nonempty train of symbols belonging to Σ; δ, ρ, $\alpha \in \Delta$; n is an integer, $1 \leq n \leq N$, where N is the total number of rules specifying the translator T; M is a train consisting of a number of integers m, $1 \leq m \leq N$; β denotes the stack.

The execution of the algorithm T uses part of the store consisting of a register ψ which contains at any moment the number of the rule in the grammar being scanned and three stacks C, B, D. The stack C is used in the translation of pairs of symbols; the stack B is used in processing the symbols the semantics of which are defined by the context on the right; the stack D is used to process symbols which make it possible to predict the possible paths of the grammatical analysis. Before execution of the algorithm the top of each stack contains the symbol $\emptyset$ and the register ψ contains 1.

The semantic equivalent for an input word (which is assumed to be terminated by a marker symbol) is constructed by a single symbol-by-symbol scan from left to right. In general the processing of each input symbol causes a review of the chain of rules of the grammar until a rule of type (1) which can be applied is detected or until syntax error is detected in the input word. The criterion that the latter has been detected is an attempt to read from empty stacks or to execute a rule of type (3); additionally, a criterion for a syntax error is that there is at least one nonempty stack when the algorithm has finished processing the marker symbol at the end of the input word.

Translated from Kibernetika, No. 2, pp. 41-44, March-April, 1972. Original article submitted December 31, 1971.

A rule of type (1) cannot be implemented if and only if: 1) the symbol in the input word being scanned does not coincide with any of the symbols in the train A; 2) it does coincide, $s=2$, but ν does not coincide with the symbol in the top of stack C.

When a rule which cannot be implemented is reviewed, the contents of ψ are increased by unity and the next rule is reviewed, its number now being contained in ψ.

The implementation of each rule of the form (1) consists in defining the semantic reaction (depending on the symbols δ and ρ) and in defining the rule for its successor.

The Case $s = 0$

1. Definition of the Semantic Reaction

If $\delta = \emptyset \wedge \rho = \emptyset$ (δ is the symbol $\emptyset$, but ρ is not), ρ is added to the output chain, and its address in the output chain is recorded in the top of stack B for possible redefinition in the light of the context on the right. The symbol being scanned is provisionally defined in this case.

If $\delta \neq \emptyset \wedge \rho \neq \emptyset$, then ρ is sent to the output chain at an address read from the top of stack B. This situation corresponds to the redefinition of a previously provisionally defined symbol.

If $\delta = \emptyset \wedge \rho = \emptyset$ there is no reaction associated with the construction of an output word (this corresponds to the processing of symbols which do not have any semantic connotation, for example, comments).

We note that the interpretation of rules with $s = 0$ does not involve the symbol ν and thus in such rules we can omit $w_0(\nu)$, which we do in the example we give.

2. Definition of the Successor Rule

If $M \neq \emptyset$, all its elements are sent to stack D. If also $n = \emptyset$, the successor rule is defined by increasing the contents of ψ by 1; otherwise it is defined by storing n in that register.

If $M = \emptyset$, then when $n \neq \emptyset$ the successor rule is defined by storing n in ψ; when $n = \emptyset$, if the stacks C, B, D are empty, the analysis has been completed successfully; if at least one of the stacks is not empty, this indicates that there is a syntax error.

In the case $s = 1$, in addition to the actions taken when $s = 0$, the symbol ν is stored in the top of stack C.

In the case $s = 2$, in addition to the actions taken when $s = 0$, the symbol at the top of C is unstacked (as already remarked, if it does not coincide with the symbol ν in the rule, the latter, by definition, is inapplicable).

Implementation of a rule of the form (2) with $\beta = D$ consists in taking the symbol from the top of stack D and storing it in ψ (this defines the number of the successor rule for the symbol being analyzed). If this symbol is $\emptyset$, the algorithm terminates with a message that there has been a syntax error.

In the case of the rule STACK B, the symbol at the top of stack B is unstacked (the hypothetical redefinition does not take place).

Implementation of a rule of the form (3) consists in the output of a message that there is a syntax error the type of which is defined by the subprogram α.

It should be noted that in addition to the definition of semantic equivalents, the interpretation of the rules of the grammar ensures complete syntactic control by detecting all the symbols of the input word which, according to the syntax of the language, are not compatible with the subchain preceding them (with the context on the left).

As an example, let us discuss a fragment of the description of a grammar of a machine language for a simple arithmetic expression in ALGOL 60 [2] in which:

⟨arithmetic expression⟩ ::= ⟨simple arithmetic expression⟩ X

⟨simple arithmetic expression⟩ ::= ⟨term⟩ | ⟨adding operator⟩ ⟨term⟩ | ⟨simple arithmetic expression⟩ ⟨adding operator⟩ ⟨term⟩

⟨adding operator⟩ ::= + | −

⟨term⟩ ::= ⟨factor⟩ | ⟨term⟩ ⟨multiplying operator⟩ | ⟨factor⟩

⟨factor⟩ ::= ⟨primary expression⟩

⟨ multiplying operator⟩ ::= × | /

⟨primary expression⟩ ::= ⟨number⟩ | (⟨arithmetic expression⟩) | ⟨variable⟩

⟨variable⟩ :: = ⟨identifier⟩ | ⟨identifier⟩ [⟨subscript list⟩]

⟨subscript list⟩ ::= ⟨simple arithmetic expression⟩ | ⟨subscript list⟩, ⟨simple arithmetic expression⟩

⟨identifier⟩ :: = ⟨letter⟩ | ⟨identifier⟩ ⟨letter⟩ | ⟨identifier⟩ ⟨digit⟩

⟨number⟩ ::= ⟨digit⟩ | ⟨number⟩ ⟨digit⟩

Let us denote trains of all letters and digits respectively by L and K; a train of +, − by $\oplus$, and a train of ×, / by $\otimes$.

In our example the machine grammar of the language is described by the following 48 rules, where $\Sigma = \{L, K, \oplus, \otimes, , [,], (,), X\}$.

1. $\oplus \rightarrow P_{13}*\emptyset *2*\emptyset$
2. $L \rightarrow \emptyset *P_7*6*12$
3. $K \rightarrow P_9*\emptyset *10*12.$
4. $(\xrightarrow{W_1(\nu_1)} P_3*\emptyset *16*25$
5. SE
6. $L \rightarrow P_8*\emptyset *6*\emptyset$
7. $K \rightarrow P_8*\emptyset *6*\emptyset$
8. $[\xrightarrow{W_1(\nu_1)} P_5*P_{11}*32*45$
9. STACK B*D
10. $K \rightarrow P_{10}*\emptyset *10*\emptyset$
11. STACK D
12. $\otimes \rightarrow P_2*\emptyset *2*\emptyset$
13. $\oplus \rightarrow P_1*\emptyset *2*\emptyset$
14. $X \rightarrow P_{14}*\emptyset *\emptyset *\emptyset$
15. SE
16. $\oplus \rightarrow P_{13}*\emptyset *17*\emptyset$
17. $L \rightarrow \emptyset *P_7*21*29$
18. $K \rightarrow P_9*\emptyset *10*29$
19. $(\xrightarrow{W_1(\nu_2)} P_3*\emptyset *16*25$
20. SE
21. $L \rightarrow P_8*\emptyset *21*\emptyset$
22. $K \rightarrow P_8*\emptyset *21*\emptyset$
23. $[\xrightarrow{W_1(\nu_2)} P_5*P_{11}*32*45$
24. STACK B*D
25. $)\xrightarrow{W_2(\nu_1)} P_4*\emptyset *12*\emptyset$
26. $)\xrightarrow{W_2(\nu_2)} P_4*\emptyset *29*\emptyset$
27. $)\xrightarrow{W_2(\nu_3)} P_4*\emptyset *41*\emptyset$

28. STACK D

29. $\oplus \rightarrow P_1 * \emptyset * 17 * \emptyset$

30. $\otimes \rightarrow P_2 * \emptyset * 17 * \emptyset$

31. STACK D

32. $\oplus \rightarrow P_{13} * \emptyset * 33 * \emptyset$

33. $L \rightarrow \emptyset * P_7 * 37 * 41$

34. $K \rightarrow P_9 * \emptyset * 10 * 41$

35. $(\xrightarrow{W_1(\nu_3)} P_3 * \emptyset * 16 * 25$

36. SE

37. $L \rightarrow P_8 * \emptyset * 37 * \emptyset$

38. $K \rightarrow P_8 * \emptyset * 37 * \emptyset$

39. $[\xrightarrow{W_1(\nu_3)} P_5 * P_{11} * 32 * 45$

40. STACK B*D

41. $\oplus \rightarrow P_1 * \emptyset * 33 * \emptyset$

42. $\otimes \rightarrow P_2 * \emptyset * 33 * \emptyset$

43. $, \rightarrow P_{12} * \emptyset * 32 * \emptyset$

44. STACK D

45. $]\xrightarrow{W_2(\nu_1)} P_6 * \emptyset * 12 * \emptyset$

46. $]\xrightarrow{W_2(\nu_2)} P_6 * \emptyset * 29 * \emptyset$

47. $]\xrightarrow{W_2(\nu_3)} P_6 * \emptyset * 41 * \emptyset$

48. STACK D

There are 14 subprograms for the semantic definitions in this grammar, each of which executes some function of the translation of the current symbol in the input sequence: P_1 and P_2 translate the adding and multiplying arithmetic operations; P_3 and P_4, P_5 and P_6 process left and right parentheses and brackets; P_7 and P_8 process the first letter and the subsequent symbols of an identifier; P_9 and P_{10} process the first and subsequent digits of a number; P_{11} processes the first letter in a subscripted variable; P_{12} processes a comma; P_{13} and P_{14} process unary operations and the end of an expression.

Let us define for our example grammatical correctness (incorrectness) and the semantic definition of the words A, A1 (examples 1, 2) in the language under consideration in accordance with the above grammar.

Example 1. As a result of the analysis of the input word

$$A : xyz \times (F2[5] - x) + 1X$$

we obtain the following chain of names of semantic subprograms, over the symbols of which we write the corresponding symbols of the input chain (under the names of the semantic subprograms we write the numbers of the rules of the grammar which apply):

x	y	z	$\times$	(	F	2	[	5	]	—	x	)	+	1	X
P_7	P_8	P_8	P_2	P_3	P_7	P_8	P_5	P_9	P_6	P_1	P_7	P_4	P_1	P_9	P_{14}
					↑	P_{11}									
2	6	6	9	4	17	22	23	34	11	29	17	21	13	3	11
			12						44			31			14
									46			31			
												25			

The names of the semantic subprograms which could be subjected to redefinition by the context on the right are underlined; in this example the second of the three underlined names was redefined, as indicated by the arrow on which is indicated the redefined name P_{11}.

When we execute this chain of subprograms, we obtain the following sequence of commands for some hypothetical single address machine:

$$\begin{aligned} &\rightarrow F\ 2\ [5] \\ &-\ x \\ &\times\ xyz \\ &+\ 1, \end{aligned}$$

where F2[5] corresponds to the address of the fifth element of the array F2, the address of which and the address of the variables x and xyz are defined in a special table previously obtained by translating the declarations of the program.

Example 2. For the word

$$A1 : ((a + 1)\,X$$

in a similar manner we obtain

(	(	a	+	1	)	X
P_3	P_3	P_7	P_1	P_9	P_4	P_{14}
4	19	17	24	18	11	28
			29		29	31
					25	

The last rule executed in the analysis of this example is rule 31, which results in an attempt to read from an empty stack D, thus indicating an error in the input word.

Practical Implementation. The above method of analysis was implemented in translators from the language D-ALGAMS [3] and the data processing language VM-COMPILER on the Dnepr-2 computer and also in the construction of an interactive debugging system for the same machine [4].

The machine grammar of the language D-ALGAMS is specified by 1200 rules and occupies 4800 bytes; the semantics of the language is defined by 221 subprograms. The machine grammar of the input language VM-COMPILER occupies 300_8 lines; the semantics of the language is defined by 108 subprograms.

A universal algorithm for the analysis of languages which can be specified by machine grammars for a computer comprises 500 machine words and runs on the Dnepr-2 computer at about 400 symbols per second, which corresponds to about 70 commands for every input symbol.

As experience in the construction of programming systems shows, the application of the above method of analysis of languages makes possible a significant reduction in the time to build such systems. Thus, for example, by using the method of a machine grammar, the VM-COMPILER was developed and put into operation in three months by three specialists.

LITERATURE CITED

1. I. V. Vel'bitskii and E. L. Yushchenko, "A metalanguage for syntactical analysis and control," Kibernetika, No. 2, Kiev (1970).
2. A. P. Ershov (editor), The Algorithmic Language ALGOL 60, a Review [Russian translation], Moscow (1965).
3. E. M. Lavrishcheva, G. A. Mos'pan, L. G. Usenko, E. L. Yushchenko, and V. A. Yaffe, "A translator for D-ALGAMS on the Dnepr-2," Trudy I Nauch.-Tekh. Konferen. SKB MMS, Kiev (1970).
4. E. M. Lavrishcheva, "Syntax-driven interactive program debugging," Trudy II Nauch.-Tekh. Konferen. SKB MMS IK AN UkrSSR, Kiev (1971).