KYBERNETIKA ČÍSLO 5, ROČNÍK 4/1968

Contribution to Deterministic Top-Down Analysis of Context-free Languages

KAREL ČULÍK II

In the present paper a generalization of $LL(k)$-grammars is given, the notion of the switching function for such grammars is introduced and the model of the Parsing Machine using the switching function is given.

We introduce the necessary notions and notation, mainly according to D. E. Knuth: An alphabet X is a finite nonempty set of symbols, and X^* denotes the set of all strings on the alphabet X. The length of a string u is denoted by $|u|$.

A context free grammar is a 4-tuple (T, N, P, S) where T, N are disjoint alphabets called terminal and nonterminal alphabets respectively; P is a finite nonempty set of productions. A production is a pair denoted by $A \rightarrow u$, where $A \in N$, $u \in (N \cup T)^*$; $S \in N$ is an initial symbol.

Let $G = (T, N, P, S)$ be a context-free grammar. For $u, v \in (T \cup N)^*$ let us write $u \Rightarrow v$ if there exist strings $x, y, w \in (T \cup N)^*$, such that $u = xAy$, $v = xwy$ and $A \rightarrow w \in P$. If $x \in T^*$ we write $u \overset{L}{\Rightarrow} v$, if $y \in T^*$ we write $u \overset{R}{\Rightarrow} v$.

The reflexive transitive completion of relation $\Rightarrow$ is denoted by $\Rightarrow^*$ and the transitive completion of $\Rightarrow$ is denoted by $\Rightarrow^+$. Similarly $\overset{L}{\Rightarrow}^*$, $\overset{L}{\Rightarrow}^+$, $\overset{R}{\Rightarrow}^*$, $\overset{R}{\Rightarrow}^+$. The set $L(G) = \{u \in T^* : S \Rightarrow^+ u\}$ is called a context-free language generated by G. If $w \in (T \cup N)^*$ we write $L(w) = \{u \in T^* : w \Rightarrow^* u\}$.

Let $u_0, u_1, \ldots, u_r$ be a sequence of strings;

(i) if $u_i \Rightarrow u_{i+1}$ $(i = 0, 1, \ldots, r - 1)$ then the sequence is called the derivation of u_r (from u_0);

(ii) if $u_i \overset{L}{\Rightarrow} u_{i+1}$ $(i = 0, 1, \ldots, r - 1)$ then the sequence is called the left-most derivation of u_r (from u_0);

(iii) if $u_i \overset{R}{\Rightarrow} u_{i+1}$ $(i = 0, 1, \ldots, r - 1)$ then the sequence is called the right-most derivation of u_r (from u_0).

A grammar G is said to be ambiguous if there is some word in $L(G)$ generated by two different left-most derivations (from S). A grammar which is not ambiguous is said to be unambiguous. The nonterminal symbol A is said to be left recursive if there exists $u \in (T \cup N)^*$ such that $A \Rightarrow^+ Au$.

For $A, B \in N$ it is said A depends on B if there exist $u, v \in (T \cup N)^*$ such that $A \Rightarrow^+ uBv$.

A nonterminal symbol A is called useless if either $L(A) = \emptyset$ or if S does not depend on A.

If k is a nonnegative integer and u is a string, we define:

$k : u$ is the initial substring of the k characters of u if the length of u is greater or equal to k;

$k : u$ is u if the length of u is less than k.

The general problem of syntactic analysis is: A given grammar $G = (T, N, P, S)$ and a string $u \in T^*$, determine whether or not $u \in L(G)$. If so, find all its syntactic structures.

The bottom-up method attacks this problem by step by step "reducing" the given string u by reductions which are the opposite of productions. If the bottom-up analysis is left to right one then the left-most possible reduction is applied at each step. This process continues until we reduce everything to S or show that this reduction would be impossible.

The top-down left to right method starts with S, and attempts to reach the left-most derivation of the string u. At each step we must decide which production is to be applied to the left-most nonterminal symbol.

There are various "back-up" procedures for both bottom-up and top-down analysis because we must reconsider some alternatives of the derivation sequence that later prove to be incorrect. For practical purposes such cases are very important when the syntactic analysis proceeds without backing up. Such procedures are called deterministic analysis methods.

D. E. Knuth [1] and Lewis and Stearns [3] introduced classes of grammars which allowed deterministic analysis.

The $LR(k)$ grammars for bottom-up left to right deterministic analysis are defined in [2] as follows:

A context-free grammar is $LR(k)$ if the following condition holds for all u_1 and u_1' in $(N \cup T)^*$, all u_2, u_2', u_3, and u_3' in T^* and all A, A' in N:

$$S \overset{R}{\Rightarrow} u_1 A u_3 \overset{R}{\Rightarrow} u_1 u_2 u_3 ,$$

$$S \overset{R}{\Rightarrow}{}^* u_1' A' u_3' \overset{R}{\Rightarrow} u_1' u_2' u_3'$$

and

$$(|u_1 u_2| + k) : u_1 u_2 u_3 = (|u_1 u_2| + k) : u_1' u_2' u_3'$$

implies that

$$u_1 = u_1' , \quad A = A' , \quad \text{and} \quad u_2 = u_2' .$$

$LL(k)$ grammars for top-down left to right deterministic analysis are defined in [2] and [3].

Definition. A context-free grammar is $LL(k)$ if the following condition holds for all u_1, u_4, u_4' in T^* and all u_2, u_3, u_2', u_3' in $(N \cup T)^*$:

$$S \overset{L}{\Rightarrow}{}^* u_1Au_3 \overset{L}{\Rightarrow} u_1u_2u_3 \overset{L}{\Rightarrow}{}^* u_1u_4 ,$$

$$S \overset{L}{\Rightarrow}{}^* u_1Au_3' \overset{L}{\Rightarrow} u_1u_2'u_3' \overset{L}{\Rightarrow}{}^* u_1u_4'$$

and

$$k : u_4 = k : u_4'$$

implies that

$$u_2 = u_2' .$$

D. E. Knuth [2] gives some comparison of top-down and bottom-up deterministic analysis:

Bottom-up analysis can deterministically parse more general languages than top-down analysis for the class of $LL(k)$-grammars is proper subset of the class of $LR(k)$-grammars. On the other hand providing top-down analysis in $LL(k)$-grammar we have a great advantage, since we know what production is being used before we actually process its components. The foreknowledge can be extremely important in practice.

The aim of this paper is a generalisation of $LL(k)$-grammars which seems to be unnecessary restrictive for deterministic analysis. We also give no-backup working Parsing Machine corresponding to them.

Definition. A context free grammar $G = (T, N, P, S)$ is said $LL(f)$ if for function f (from T^* to arbitrary range D) the following condition holds for all

(1) u_1, u_4, u_4' in T^* and all u_2, u_3, u_2', u_3' in $(N \cup T)^*$:

(2) $$S \overset{L}{\Rightarrow}{}^* u_1Au_3 \overset{L}{\Rightarrow} u_1u_2u_3 \overset{L}{\Rightarrow}{}^* u_1u_4 ,$$

(3) $$S \overset{L}{\Rightarrow}{}^* u_1Au_3' \overset{L}{\Rightarrow} u_1u_2'u_3' \overset{L}{\Rightarrow}{}^* u_1u_4'$$

and

$$f(u_4) = f(u_4')$$

implies that

$$u_2 = u_2' .$$

Function f is called distinctive function for grammar G.

Note 1. Setting $f(u) = k : u$ we get the $LL(k)$ grammars.

Theorem 1. *There exists a distinctive function f for grammar G if and only if the grammar G is unambiguous and has no nonuseless left-recursive nonterminal symbols.*

Proof. 1. Let us assume that f is a distinctive function for grammar G.

a) Let G be ambiguous. Then threre exists (1) such that (2) and (3) hold, $u_4 = u_4'$ and $u_2 \neq u_2'$. For every f follows that $f(u_4) = f(u_4')$ holds and we have a contradiction with the definition of distinctive function.

b) Let grammar G have the nonuseless left-recursive nonterminal symbol A. Then there exists $u_1, u_4 \in T^*$, u_2, u_2', $u_3 : u_3 \in (N \cup T)^*$, $u_2 \neq u_3'$ for which

$$S \overset{L}{\Rightarrow}{}^* u_1Au_3 \overset{L}{\Rightarrow} u_1u_2u_3 \overset{L}{\Rightarrow}{}^* u_1Au_3' \overset{L}{\Rightarrow} u_1u_2'u_3' \overset{L}{\Rightarrow}{}^* u_1u_4$$

holds. This is a contradiction with the assumption that there exists a distinctive function for G.

2. Let G be an unambiguous grammar which has no nonuseless left-recursive nonterminal symbols.

Let us set $f(u) = u$ for all $u \in T^*$. Let us assume that there are such (1) that (2) and (3) are valid, $u_4 = u_4'$ and $u_2 \neq u_2'$.

Because of the unambiguity of grammar G either

$$S \overset{L}{\Rightarrow}{}^* u_1Au_3 \overset{L}{\Rightarrow} u_1u_2u_3 \overset{L}{\Rightarrow}{}^* u_1Au_3' \overset{L}{\Rightarrow} u_1u_2'u_3' \overset{L}{\Rightarrow}{}^* u_1u_4$$

or

$$S \overset{L}{\Rightarrow}{}^* u_1Au_3' \overset{L}{\Rightarrow} u_1u_2'u_3' \overset{L}{\Rightarrow}{}^* u_1Au_3 \overset{L}{\Rightarrow} u_1u_2u_3 \overset{L}{\Rightarrow}{}^* u_1u_4$$

is valid and consequently A is nonuseless left-recursive and it is a contradiction.

Definition 2. Let the rules of grammar G be rewritten in the form $A \to w_1 \mid w_2 \mid \dots$ $\dots \mid w_r$. (All the productions with the same left side are substituted for one generalized production). The integer-value function $F(u, A, v)$ is said to be a switching function for grammar G if it is defined for all u, v in T^* and A in N such, that

$$(4) \qquad S \overset{L}{\Rightarrow}{}^* uAu' \overset{L}{\Rightarrow} uw_iu' \overset{L}{\Rightarrow}{}^* uv$$

where $u' \in (T \cup N)^*$ and $F(u, A, v) = i$ is valid.

Note 2. The significance of switching function for top-down left-to-right analysis is obvious.

Theorem 2. *If f is a distinctive function for grammar G then there exists a function g (from $T^* \times N \times D$ to I, where D is the range of values of the function f*

and I denotes the set of natural numbers) such that the composed function F defined as

$$F(u, A, v) = g(u, A, f(v))$$

is the switching function for grammar G.

Proof. The function g is defined as follows. Let (4) be valid and $f(v) = y$ then we set $g(u, A, y) = i$. Let besides (4)

$$(4') \qquad S \overset{L}{\Rightarrow}{}^* uAu'' \overset{L}{\Rightarrow} uw_ju'' \overset{L}{\Rightarrow}{}^* uv'$$

be valid and $f(v') = f(v) = y$. Then from the fact that f is a distinctive function it follows $w_i = w_j$. Therefore f is chosen uniquely and it is obvious that f is a switching function for grammar G.

Definition 3. A context-free grammar $G = (T, N, P, S)$ is said $LLS(f)$ if for the function f (from T^* to arbitrary D) the following condition holds for all

$$(1') \qquad u_1, u_1', u_4, u_4' \in T^* \quad \text{and all} \quad u_2, u_2', u_3', u_3' \in (N \cup T)^* ,$$

$$(2') \qquad S \overset{L}{\Rightarrow}{}^* u_1Au_3 \overset{L}{\Rightarrow} u_1u_2u_3 \overset{L}{\Rightarrow}{}^* u_1u_4 ,$$

$$(3') \qquad S \overset{L}{\Rightarrow}{}^* u_1'Au_3' \overset{L}{\Rightarrow} u_1'u_2'u_3' \overset{L}{\Rightarrow}{}^* u_1'u_4'$$

and

$$f(u_4) = f(u_4')$$

implies that

$$u_2 = u_2' .$$

Function f is called a strongly distinctive function for grammar G.

Example 1. *Let* $G = (\{a, b, c, d\}, \{S, A\}, \{S \to cAb \mid dA, A \to a \mid ab\}, S)$.
Owing to the fact, that

$$S \overset{L}{\Rightarrow} cAb \overset{L}{\Rightarrow} cab$$

$$S \overset{L}{\Rightarrow} dA \overset{L}{\Rightarrow} dab ,$$

no strongly distinctive function exists for grammar G. On the other hand it is obvious that G is $LL(3)$.

Definition 4. Let the productions of the grammar G be rewritten in the form $A \to w_1 \mid w_2 \mid \ldots \mid w_r$. The integer-value function $F(A, v)$ is said to be a strongly switching function for the grammar G if it is defined for all $v \in T^*$ and $A \in N$ such that

$$S \overset{L}{\Rightarrow}{}^* uAu' \overset{L}{\Rightarrow} uw_iu' \overset{L}{\Rightarrow}{}^* uv$$

where u' is in $(T \cup N)^*$ and $F(A, v) = i$ is valid.

Theorem 3. *If f is a strongly distinctive function for the grammar G then there exists a function g (from $N \times D$ to I, where D is the range of values of the function f and I denotes the set of natural numbers) such that the composed function F defined as*

$$F(A, v) = g(A, f(v))$$

is a strongly switching function for the grammar G.

Proof. It is analogous to the Theorem 2.

A classification of a context-free languages according to the necessary complexity of (strongly) distinctive function of their grammars can be introduced. For instance:

1. $LL(k)$-languages are languages generated by grammars for which $f(u) = k : u$ is the distinctive function.
2. Languages generated by grammars for which a distinctive function is sequential. Sequential function is the function which is realized by a finite state sequential machine.

Let us assume that we have a procedure computing the value of (strongly) switching function $F(u, A, v)$ $(F(A, v))$ (for some grammar G, productions of which are written in the form $A \to w_1|w_2| \dots w_r$. Then we can modify the Knuth's Parsing Machine into a simple form which works no back-up.

The Parsing Machine is an abstract machine which is made to analyze strings over a certain alphabet. Is works character per character, according to a program. A Parsing Machine program is a sequence of instructions. One type of instructions are procedures calling each other recursively. Each such procedure attemps to find an occurence of a particular syntactic type in the input.

The Parsing Machine has to decide if a given input is in the language or not and to give the phrase marker of the string. The phrase marker will be described so that every syntactic unit in the string will be closed in brackets and under the opening bracket will be written the corresponding letter of the nonterminal alphabet N.

Let the input string be $s_1s_2 \dots s_n$, and let s_h be the "current" character being scanned by the machine.

A program is written using four types of instructions:

Type 1: A letter of the terminal alphabet;
Type 2: A letter of the nonterminal alphabet;
Type 3: *RETURN*;
Type 4: *STOP*.

Writing a program we put symbolic locations to the left of some instruction. They are written as nonterminal letters with an integer index. A program is created of the segment

```
START  S
       STOP
```

and one another segment of the form

```
$A_1$ | $q_{1,1}$
      | $q_{1,2}$
      | ,
      | ,
      | ,
      | $q_{1,m_1}$
      | RETURN
$A_2$ | $q_{2,1}$
      | $q_{2,2}$
      | .
      | .
      | .
      | $q_{2,m_2}$
      | RETURN
  .   | .
  .   | .
  .   | .
$A_r$ | $q_{r,1}$
      | $q_{r,2}$
      | .
      | .
      | .
      | $q_{r,m_r}$
      | RETURN
```

for each production $A \to w_1|w_2| \dots |w_r$
where $w_i = q_{i,1}q_{i,2} \dots q_{i,m_i}$ $(i = 1, 2, \dots, r)$, $q_{i,j} \in N \cup T$.

The program starts his work on the location *START* and the effects of instructions are following. (In description some notations of ALGOL 60 are used.)

```
Type 1. $(a \in T)$:   if $s_h = a$ then begin $h := h + 1$;
                                    outsymbol $(a)$;
                                    go to next location
                              end
                        else ERROR
```

Type 2. $(A \in N)$: *output* $(\underset{A}{\sqsubset})$ and call on the procedure which starts in location $A_{F(u,A,v)}$ recursively.

Type 3. $(RETURN)$ The end of the call of procedure, *outsymbol* $(\sqsupset)$.

Type 4. $(STOP)$ The end of work of the program. Analyzed string is in $L(G)$.

Note 3. Using a "stack" we can describe the meaning of instructions of types 2, 3 in more details.

Type 2. $(A \in N)$ h is not changed: *outsymbol* $(\underset{A}{\sqsubset})$; put current location increased by one into a stack; **go to** $A_{F(u,A,v)}$.

Type 3. $(RETURN)$ h is not changed; *outsymbol* $(\sqsupset)$; popped off top location from stack and go to the location that was popped off.

Note 4. For strongly distinctive function it is the only difference that F (u, A, v) doesn't depend on u.

Example 2. Let us write the program for the Parsing Machine performing analysis of simple Boolean expressions described by the grammar:

$$G = (T, \{V, E, R, B, X, Y, P, Q\}, P, B)$$

where

$$T = \{a, b, c, +, -, *, <, >, \neg, \vee, \wedge\}$$

nad P consists of productions

$$\begin{aligned}
&V \to a \mid b \mid c\,,\\
&X \to + \mid - \mid \times\,,\\
&E \to V \mid VXE \mid (E)\,,\\
&Y \to < \mid >\,,\\
&R \to EYE\,,\\
&P \to V \mid R \mid (B)\,,\\
&Q \to P \mid \neg\, P\,,\\
&Z \to \wedge \mid \vee\,,\\
&B \to Q \mid QZB\,.
\end{aligned}$$

Using conditional expressions of ALGOL-60 with non-ALGOL conditions we describe the strongly switching function F for the grammar G. In these conditions the symbol $=$ is used for comparing the two strings of terminal symbols; mind the symbol (among them.

```
$F(V, v) =$ if $v = av', v' \in T^*$ then 1
                  else if $v = bv', v' \in T^*$ then 2
                              else if $v = cv', v' \in T^*$ then 3
                                          else $ERROR$;
$F(X, v) =$ if $v = +v', v' \in T^*$ then 1
                  else if $v = -v', v' \in T^*$ then 2
                              else if $v = \times\ v', v' \in T^*$ then 3
                                          else $ERROR$;
$F(E, v) =$ if $v = (v', v' \in T$ then 3
                  else if $v = \xi_1\xi_2 v', \xi_2 \in \{+, -, \times\}, \xi_1 \in \{a, b, c\}, v' \in T$ then 2
                                          else 1;
```

$F(Y, v) =$ **if** $v = \, <v', v' \in T^*$ **then** 1
else if $v = \, >v', v' \in T^*$ **then** 2
else *ERROR*;

$F(R, v) = 1$;

$F(P, v) =$ **if** $v = (v', v' \in T^*$ **then** 3
else if $v = y\xi, y \in (T - \{\neg, \vee, \wedge, >, <\})^*, \xi \in \{<, >\}$ **then** 2
else 1;

$F(Q, v) =$ **if** $v = \neg v', v' \in T^*$ **then** 2 **else** 1;

$F(Z, v) =$ **if** $v = \wedge v', v' \in T^*$ **then** 1;
else if $v = \vee v', v' \in T^*$ **then** 2
else *ERROR*;

$F(B, v) =$ **if** $v = y\xi, y \in (T - \{\wedge, \vee\})^*, \xi \in \{\wedge, \vee\}$ **then** 2 **else** 1

Program for the Parsing Machine is shown in Table 1.

Table 1.

Location	Instruction	Location	Instruction	Location	Instruction	Location	Instruction
START	*B*		*RETURN*		*E*	Z_1	$\wedge$
	STOP	E_2	*V*		*RETURN*		*RETURN*
V_1	*a*		*X*	P_1	*V*	Z_2	$\vee$
	RETURN		*E*		*RETURN*		*RETURN*
V_2	*b*		*RETURN*	P_2	*R*	B_1	*Q*
	RETURN	E_3	(		*RETURN*		*RETURN*
V_3	*c*		*E*	P_3	(	B_2	*Q*
	RETURN		)		*B*		*Z*
X_1	+		*RETURN*		)		*B*
	RETURN	Y_1	<		*RETURN*		*RETURN*
X_2	−		*RETURN*	Q_1	*P*		
	RETURN	Y_2	>		*RETURN*		
X_3	×		*RETURN*	Q_2	$\neg$		
	RETURN	R_1	*E*		*P*		
E_1	*V*		*Y*		*RETURN*		

It is natural that in the practical cases we try to choose such grammars for which the calculation of the distinctive function is simple, i.e. not taking much of both time and storage.

In the following paper we try to give a modification of the Parsing Machine determined for self-correcting of some syntactical errors and good diagnostic of others. We will apply it in analysis of preprocessed ALGOL-60 programs.

(Received March 12th, 1968)

REFERENCES

[1] D. E. Knuth: On the translation of languages from left to right. Information and Control *8* (1965), 607—639.

[2] D. E. Knuth: Top-down syntax analysis. Textbook of International Summer School on Computer Programming Copenhagen, Denmark, 1967.

[3] P. M. Lewis and R. E. Stearns: Syntax-Directed Transduction. Journal of the ACM *15* (1968), to appear.

VÝTAH

Příspěvek k deterministické analýze bezkontextových jazyků shora

KAREL ČULÍK II

Práce se zabývá analýzou bezkontextových jazyků a to analýzou shora, zleva doprava. Jsou zobecněny Knuthovy $LL(k)$-gramatiky a zavedeny pojmy (silně) rozlišovací funkce a (silně) rozvětvovací funkce. Jsou ukázány nutné a postačující podmínky k tomu, aby existovala rozlišovací a rozvětvovací funkce pro danou gramatiku. Dále je pro gramatiky, pro které existuje rozvětvovací funkce, dána modifikace Knuthova analyzátoru, který pracuje bez vracení. Je uveden příklad programu takového analyzátoru pro gramatiku popisující jednoduché Booleovské výrazy.

Dr Karel Čulík, CSc., Centrum numerické matematiky KU, Malostranské nám. 25, Praha 1.