KYBERNETIKA ČÍSLO 3, ROČNÍK 4/1968

A Contribution to the Top-to-Bottom Recognizer Rehabilitation

JIŘÍ KOPŘIVA

The method of parsing of the deterministic context-free languages described by D. E. Knuth in [1] is interpreted there only as a bottom-to-top analysis. It is shown here that this method also admits to be treated as a kind of the top to-bottom parsing.

1. INTRODUCTION

As shown in [1], *deterministic context-free languages* (i. e. sets of such strings over a finite alphabet that are recognized by deterministic push-down automata) coincide with the languages generated by *grammars translatable from left to right with bound k* (briefly "LR(k) *grammars*"). That means if we interpret the method of parsing described in [1] as BT-analysis, each reduction (i. e. the replacing of an occurrence of the right-hand side of a production by its left-hand side) can be performed *once for all* on the basis of certain information on the string to the left of this occurence and on the k terminal characters to its right. In order that the necessary information on the string to the left may be recorded and also the k terminal characters to the right may be compared, certain sets $\mathscr{S}$ and $\mathscr{S}'$ of so called *states* are built up during the process. These states contain also the information on all productions (i. e. rules of the considered grammar) we are already working on and we might begin to work on. *From the point of view of TB-analysis, it is a record of productions such that some of them might label the stage by stage arising vertices of the phrase marker. A performed reduction means then the exhaustion of such part of the (terminal) input string that is a "value" of the corresponding intermediate symbol and return to the superior level* (cf. for instance the description of TB-analysis in [2]). *Thus the succesive changes of the analysed string (caused by the reductions) need not be performed. They may be replaced by a suitable record concerning the productions whose use either has been found or will be found later. Therefore, the process loses the character of BT-analysis and becomes a kind of TB-analysis.*

In what follows a detailed explanation of this approach will be given and the algorithm of the corresponding TB-analysis will be formulated.

A *phrase marker* is a rooted oriented tree, each vertex of which is labelled by a production. Let two branches, having their final vertices labelled by productions (rules) r_2, r_3, have their common initial vertex labelled by r_1. Then in the derivation of the corresponding string, the right-hand sides of r_2, r_3 are substituted for two intermediates from the right-hand side of r_1. If in the graph of the tree the branch $r_1 r_2$ is situated on the left of $r_1 r_3$, then the right-hand side of r_2 is substituted for an intermediate laying in r_1 to the left of the intermediate replaced by the right-hand side of r_3.

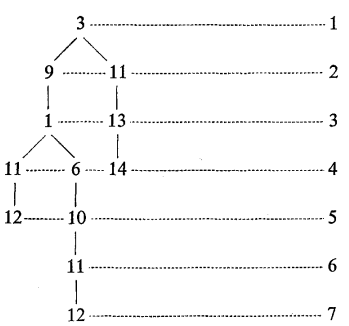
For the grammar

(1)

1. $F \to U \wedge P$,	5. $P \to U \wedge P$,	9. $U \to (F)$,	12. $L \to p$,
2. $F \to U \vee S$,	6. $P \to U$,	10. $U \to \neg U$,	13. $L \to pM$,
3. $F \to U \supset U$,	7. $S \to U \vee S$,	11. $U \to L$,	14. $M \to '$,
4. $F \to U$,	8. $S \to U$,		15. $M \to 'M$,

where F is the designated symbol (cf. [3]),

(2)



is the phrase marker of the string

(3) $$(p \wedge \neg p) \supset p' .$$

In (2), the productions are replaced by their ordinal numbers taken from (1). In what follows the set of productions is always supposed to be ordered and the corresponding ordinal number is used as the "name" of the production.

All vertices with the same distance from the root (i. e. the paths connecting them with the root have the same number of branches) build a *level*. The levels are designated by ordinal numbers beginning from the root (cf. (2)).

The phrase marker is fully described by a partially ordered set of pairs (l, r), where l is level and r production, provided that all pairs (l, r) with the same l constitute a linearly ordered set. This ordering $\prec$ reflects the position of the intermediates in the string whose phrase marker is a subtree of the considered tree such that it contains only levels $1, \ldots, l-1$. Furthermore, some further obvious conditions must hold.

The algorithm of TB-analysis described below yields a sequence of pairs (l_g, r_g), $g = 0, 1, \ldots, m$, such that for (l_i, r_i), (l_j, r_j), where $l_i = l_j$, from $i < j$ follows $(l_i, r_i) \prec (l_j, r_j)$.

The description of the analysis algorithm uses the notions and designations from [1]. For better understanding we introduce the most necessary of them here.

p-th production of the considered grammar is supposed to have the form

$$A_p \to X_{p,1} X_{p,2} \ldots X_{p,n(p)}, \quad n(p) \geqq 0 .$$

Thus A_p is an intermediate and $X_{p,i}$, $1 \leqq i \leqq n(p)$, are intermediates or terminals. The case $n(p) = 0$ corresponds to a production with empty right-hand side, i. e. $A_p \to \Lambda$. The so called "zeroth" production of the form

$$S_0 \to S \dashv^k \quad (\dashv^k \text{ means } \underbrace{\dashv \ldots \dashv}_{k\text{-times}}) \tag{4}$$

is added, where S is the designated symbol of the original grammar. The existence of k symbols to the right of the just scanned symbol of the input string in any stage of process is insured in this way. Symbol $\dashv$ is supposed to belong neither to the original terminal alphabet not to the set of intermediates.

Instead of states $[p, j; \alpha]$ from [1] two kinds of states will be introduced here.

(a) States of the 1st sort having the form

$$[t : p, j; p(t), j(t); p(t-1), j(t-1); \ldots ; p(1), j(1)] . \tag{5}$$

Here, t designates the level and p the production (or, more precisely, its ordinal number) that could be used on level t. j means that scanning the input (analysed) string and, if need be, descending at the same time to lower levels and returning to superior ones, exactly j initial symbols of the right-hand side of the p th production have been found. A *descent* to a lower level means that scanning the right-hand side of some production an intermediate was met and the future possible application of the productions having this intermediate their left-hand side was prepared. Each met terminal symbol (of the right-hand side of some production) is compared at once with the corresponding symbol of the input string. A *return* to the superior level means that the right-hand side of some production has been exhausted is this

way and therefore the *definitive decision concerning the application of certain production on certain level has been done.*

The following members $p(i)$, $j(i)$ in (5) designate also a production and subscript of a symbol of its right-hand side resp. But they are used to forming the k-letter strings which, on a suitable stage, are compared with k following up to now unused symbols of the input string. E. g. for (5), the corresponding k-letter strings are built in the following way: All the strings that can be derived from the final segment of the right-hand side of the production $p(t)$ beginning with $X_{p(t),j(t)}$ are concatenated with all the strings that can be derived from the final segment of the right-hand side of the production $p(t-1)$ beginning with $X_{p(t-1),j(t-1)}$, etc. Only initial k-letter terminal substrings of the formed strings are considered. It is obvious that the number of "double components" $p(i)$, $j(i)$ can be limited so that it corresponds to the shortest string

$$X_{p(t),j(t)}X_{p(t),j(t)+1}\cdots X_{p(t),n(p(t))}X_{p(t-1),j(t-1)}X_{p(t-1),j(t-1)+1}\cdots \tag{6}$$

such that contains exactly k symbols such that each of them is either terminal or, if it is intermediate, then all productions with this symbol on their left-hand side have a nonempty right-hand side.

(b) States of the 2nd sort having the form $[t : p, n(p); \alpha]$. Here, the first three components have the same meaning as in (a); of course, $j = n(p)$, i. e. all $n(p)$ symbols of the right-hand side of production p have been determined. α stands for k-letter terminal string.

In [1], oniy the analogues of the states of the 2nd sort are used. But, as it is obvious from the course of the process of analysis, the k-letter terminal strings need not be produced *before certain stages* of the process; *elsewhere only a record of information on from what final segments of what productions they should be built is sufficient.* It is suitable to establish the states of the 1st sort only for comparatively great k's, while for small k's it would be better to use only the states of the 2nd sort, where α is replaced by k-letter strings generated from (6).

For the reason of a brief record the sets $H_k(\sigma)$ and $H'_k(\sigma)$ are introduced. One defines $H_k(\sigma)$ to be the set of all k-letter terminal strings α ($\dashv$ is treated as a terminal here) such that string $\alpha\beta$ for some β is generable from σ with respect to the considered grammar. $H'_k(\sigma)$ means the same as $H_k(\sigma)$ except omitting all derivations that contain a step where an intermediate as the initial character is replaced by Λ.

For instance (see [1]; upper case letters S, B, C, D stand for intermediates and lower case letters c, d, e stand for terminals) in the grammar

$$S \to BC\dashv\dashv, B \to Ce, B \to \Lambda, C \to D, C \to Dc, D \to \Lambda, D \to d$$

we have

$$H_3(S) = \{\dashv\dashv, c\dashv\dashv, ce\dashv, cec, d\dashv\dashv, dce, de\dashv, dec, ded, e\dashv\dashv, ec\dashv, ed\dashv, edc\},$$

$$H'_3(S) = \{dce, de\dashv, dec, ded\}.$$

$\mathscr{S}$ and $\mathscr{S}'$ with indices (below and above) stand for the sets of states. The meaning of the other used symbols and letters will be patent from the description of the algorithm (the current member of the input analysed string is a_i). For the reason of briefness, this description (given in the following section) is written in a pseudo-algol form (e. g., it uses some kinds of ALGOL statements, integers as labels, etc.). The resulting phrase marker contains also a zeroth level on which only one (viz. the zeroth) production is used (see values l_0 and r_0 in statement 1 of the algorithm).

The considered grammar is supposed not to contain intermediates with the left recursion. A procedure removing such intermediates and then giving the syntactic structure with respect to the original grammar is described e. g. in [4].

The problem of deciding, for a given grammar $\mathfrak{G}$, whether or not there exists a $k \geqq 0$ such that $\mathfrak{G}$ is LR(k), is recursively unsolvable. This is proved in [1], where also certain methods for finding this property for a grammar, when k is given, are described. See also the note beyond the algorithm in the next section.

2. THE TB-ANALYSIS ALGORITHM

begin

1: $i := n := l_0 := r_0 := 0$; $\mathscr{S}_n := \{[0 : 0, 0]\}$;

2: $h := 0$; ${}^h\mathscr{S}'_n := \mathscr{S}_n$;

3: $h := h + 1$;

$${}^h\mathscr{S}'_n := \{[t+1 : q, 0; p, j+2; p(t), j(t); \ldots] \mid [t : p, j; p(t), j(t); \ldots] \in {}^{h-1}\mathscr{S}'_n \wedge X_{p,j+1} = A_q \wedge \neg(A_q \to \Lambda)\} \cup \{[t+1; q, 0; \alpha] \mid [t; p, j; p(t), j(t); \ldots] \in {}^{h-1}\mathscr{S}'_n \wedge X_{p,j+1} = A_q \wedge A_q \to \Lambda \wedge \alpha \in H_k(X_{p,j+2}X_{p,j+3} \ldots X_{p,n(p)}X_{p(t),j(t)}X_{p(t),j(t)+1} \cdots \ldots X_{p(t),n(p(t))} \cdots \dashv^k)\} ;$$

if ${}^h\mathscr{S}'_n = \emptyset$ **then begin** $\mathscr{S}'_n := \bigcup_{j=0}^{h-1} {}^j\mathscr{S}'_n$; **goto** 4 **end else goto** 3;

comment: *All productions we might begin to work on are prepared.*;

4: **for** $p := 1$ **step** 1 **until** s **do**

if $a_{i+1} \ldots a_{i+k} \in \{\alpha \mid [t : p, n(p); \alpha] \in \mathscr{S}'_n\}$ **then goto** 5;

comment: *The whole righthand side of the p th production exhausted — return to the superior level follows.*;

if $a_{i+1} \ldots a_{i+k} \in \{\alpha \mid [t : p, j; p(t), j(t); \ldots] \in \mathscr{S}'_n \wedge \alpha \in H'_k(X_{p,j+1}X_{p,j+2} \ldots X_{p,n(p)}X_{p(t),j(t)}X_{p(t),j(t)+1} \ldots X_{p(t),n(p(t))} \cdots \dashv^k)\}$

then goto 6 **else** *stop — analysed string does not belong to the language*;

comment: *The whole righthand side of the corresponding production has not been found — shift to the next symbol follows.*;

5: $l_g := t; r_g := p;$
if $l_g = 1 \wedge a_{i+1} = \dashv$ **then** *stop – end of analysis*;
$g := g + 1; n := n - n(p); X := A_p$; **goto** 7;

6: $i := i + 1; X := a_i;$

7: $n : n + 1;$
$\mathscr{S}_n := \{[t : p, j + 1 ; p(t), j(t); \ldots] \mid [t : p, j; p(t), j(t); \ldots] \in \mathscr{S}'_{n-1} \wedge$
$X = X_{p,j+1} \wedge j + 1 < n(p)\} \cup$
$\{[t : p, n(p); \alpha] \mid [t : p, j; p(t), j(t); \ldots] \in \mathscr{S}'_{n-1} \wedge$
$X = X_{p,n(p)} \wedge n(p) = j + 1 \wedge \alpha \in H_k(X_{p(t),j(t)} X_{p(t),j(t)+1} \cdots$
$\cdots X_{p(t),n(p(t))} \cdots \dashv^k)\}$;
comment: *The new* $\mathscr{S}_n$ *has been formed.*;
if $\mathscr{S}_n = \emptyset$ **then** *stop – analysed string does not belong to the language*
else goto 2
end

Note. The sets of the terminal strings α, which are built in statement 4 for particular p's, and the set built in the following one must all be disjoint sets, or the grammar is not LR(k). There exists only finite number of combinations we get for a given k in that way. The investigation of them is one of the methods for finding the property LR(k) for the considered grammar.

3. EXAMPLE

From the detailed investigation of the grammar (1) it follows it is LR(1). In the first place obviously a found p or $'$ causes a return (to the superior level) according to production 12 or 14 resp. or a shift to the next symbol of the input string (with subsequent return according to production 13 or 15) on the basis of knowledge of one following character of the input string (whether it is $'$ or another symbol). One symbol to the right allows also, for found U, to decide whether a return (according to one of productions 4, 6 and 8) or a shift to the next character of the input string will be performed. The latter event happens if U is followed by one symbol of $\wedge$, $\vee$, $\supset$, the former in another case. Of course, which of productions will be applied for the return, this is decided not only according to one following symbol to the right itself but also according to the knowledge of up to now investigated initial segment of the input string (i. e. according to the productions whose numbers are the second members of the corresponding states). Similarly for found strings $U \wedge P$ and $U \wedge S$. Let us show this procedure with help of the part of analysis of the string (3), which, of course, has to be written in the form

$$(p \wedge \neg p) \supset p' \dashv$$

now.

Grammar (1) contains no production with an empty right-hand side. Therefore, since $k = 1$, only states of the 1st sort having the form $[t : p, j; p', j']$, will be sufficient (besides states of the 2nd sort, of course). (We shall use the states of the 1st sort here in spite of we have $k = 1$.)

First, we get pair $(l_0, r_0) = (0, 0)$; this is a trivial result of each analysis if we use a grammar with an added zeroth production of the form (4). It is

$$\mathscr{S}_0 = \{[0; 0, 0]\}$$

and we get

$$\begin{aligned}\mathscr{S}'_0 = \mathscr{S}_0 \cup \{[1 : u, 0; 0, 2] \mid u = 1, 2, 3, 4\} \cup \\ \{[2 : u, 0; v, 2] \mid u = 9, 10, 11; v = 0, 1, 2, 3\} \cup \\ \{[3 : u, 0; v, 2] \mid u = 12, 13; v = 0, 1, 2, 3\} .\end{aligned}$$

Since $a_1 = ($, we get

$$\mathscr{S}_1 = \{[2 : 9, 1; v, 2] \mid v = 0, 1, 2, 3\}$$

and

$$\begin{aligned}\mathscr{S}'_1 = \mathscr{S}_1 \cup \{[3 : u, 0; 9, 3] \mid u = 1, 2, 3, 4\} \cup \{[4 : u, 0; v, 2] \mid u = \\ = 9, 10, 11; v = 1, 2, 3\} \cup \\ \{[4 : u, 0; 9, 3] \mid u = 9, 10, 11\} \cup \{[5 : u, 0; v, 2] \mid u = \\ = 12, 13; v = 1, 2, 3\} \cup \\ \{[5 : u, 0; 9, 3] \mid u = 12, 13\} .\end{aligned}$$

Now, from $a_2 = p$ follows

$$\begin{aligned}\mathscr{S}_2 = \{[5 : 12, 1; a] \mid a = \wedge, \vee, \supset,)\} \cup \{[5 : 13, 1; v, 2] \mid v = \\ = 1, 2, 3\} \cup \{[5 : 13, 1; 9, 3]\}\end{aligned}$$

and thus

$$\begin{aligned}\mathscr{S}'_2 = \mathscr{S}_2 \cup \{[6 : u, 0; v, 2] \mid u = 14, 15; v = 1, 2, 3\} \cup \\ \cup \{[6 : u, 0; 9, 3] \mid u = 14, 15\} .\end{aligned}$$

It is $a_3 = \wedge$ and, therefore, we get further pair $(l_1, r_1) = (5, 12)$, i. e. the definitive decision is done that the production 12 is used on the level 5. Since $A_{12} = L$, after return to $\mathscr{S}'_1$, we get the new

$$\mathscr{S}_2 = \{[4 : 11, 1; a] \mid a = \wedge, \vee, \supset,)\} ;$$

by it,

$$\mathscr{S}'_2 = \mathscr{S}_2 .$$

$a_3 = 1$ gives $(l_2, r_2) = (4,11)$, etc.

In this way, we obtain successively the sequence of pairs (including those which were found above)
(0,0), (5,12), (4,11), (7,12), (6,11), (5,10), (4,6), (3,1), (2,9), (4,14), (3,13), (2,11), (1,3).
This sequence really describes phrase marker (2), which has to be completed by

above.

I think, it will be possible to find other reasons to the fact that *there are many intrinsic connections between* TB *and* BT-*analysis* particularly if some special kinds of languages are dealt with. Another example yields standard grammar.

(Received September 11th, 1967.)

REFERENCES

[1] Knuth, Donald E.: On the Translation of Languages from Left to Right. Information and Control *8* (1965), 607—639.
[2] Floyd, R. W.: The Syntax of Programming Languages — A Survey. IEEE Transactions on Electronic Computers (Aug. 1964), 346—353.
[3] Brooker, R. A.: Top-to Bottom Parsing Rehabilitated? Comm. ACM *10* (April 1967), 2, 223—224.
[4] Kurki-Suonio, R.: On Top-to-Bottom Recognition and Left Recursion. Comm. ACM *9* (July, 1966), 7, 527—528.

VÝTAH

Příspěvek k rehabilitaci syntaktické analýzy shora

JIŘÍ KOPŘIVA

Hlavním výsledkem práce je algoritmus potvrzující možnost interpretovat způsob syntaktické analýzy deterministických bezkontextových jazyků, popsaný D. E. Knuthem v [1], jako syntaktickou analýzu shora (top-to-bottom analysis) bez návratů, tj. analýzu, při níž každé rozhodnutí o použití určitého gramatického pravidla pro přepis jednotlivých metaproměnných je definitivní.

Dr. Jiří Kopřiva CSc., Laboratoř počítacích strojů, Třída Obránců míru 21, Brno.