PARSING AS A SUBTASK OF COMPILING

Jaroslav Král
Jiří Demner
Technical University of Prague,
Institute of Computation Technique

Summary

In the last years the research of context-free languages was concentrated on the problems of syntactic analysis and recently on the problems of the integrated theory of parsing and compiling using affix (Koster [18]) or attribute grammars (Lewis [25], Bochmann [2], Knuth [16]).

In this paper the requirements on parsing methods in compilers are discussed. The advantages of top-down parsing method are summarized and a modification of the top-down method able to parse LR(k) grammars without left recursive symbols is given.

The problem of the syntactic analysis is then reduced to the problem how to insert "action symbols" (controlling the compiling) into the input string. It is shown that in such a treatment the differences between the bottom-up and top-down methods are in some sense less important. Some bottom-up methods, especially the LR(k) ones, have hidden top-down features which can be easily used up. Some modifications of bottom up methods allowing to produce top-down like parsing information are discussed.

1. Introduction

We shall assume that the reader is familiar with the grounds of the theory of formal languages and parsing (see Aho, Ullmann [1]). We shall give here our notational conventions.

The context-free grammar G is the quadruple \( G = (N, T, P, S) \). \( N \) and \( T \) are disjoint finite sets called nonterminal and terminal alphabets,
respectively. Elements of \( N (T) \) are nonterminal (terminal) symbols.

\( S \), the initial symbol, is a symbol from \( N \). \( P \) is a finite set of rules (productions) of the form \( A \rightarrow w \), where \( w \) is a sequence of symbols from \( N \) and \( T \) (i.e., a word over \( N \cup T \)) and \( A \) is from \( N \). If \( C \) is an alphabet (i.e., a finite set) then \( C^* \) is the set of all words over \( C \). The length of a word \( w \) from \( C^* \) is denoted \( |w| \). The empty string is denoted \( \lambda \). \( C^+ = C - \{ \lambda \} \). Unless there is the danger of misunderstanding we shall not distinguish the one element set \( \{ a \} \) and its one element \( a \).

If \( A \rightarrow w \) is from \( P \), \( x, y \) from \( (N \cup T)^* \), \( x = x_1 A x_2 \) and \( y = x_1 w x_2 \) then \( y \) is an immediate consequence of \( x \), written \( x \Rightarrow y \). If moreover \( x_1 \) (\( x_2 \)) is from \( T^* \) we write \( x \Rightarrow_L y \) (\( x \Rightarrow_R y \), respectively). Let \( \Rightarrow^* \), \( \Rightarrow_L^* \), \( \Rightarrow_R^* \) be the reflexive transitive closures of \( \Rightarrow \), \( \Rightarrow_L \), \( \Rightarrow_R \), respectively. A (left-most) derivation over \( G \) is the sequence of words \( x_0, x_1, x_2, \ldots, x_n \) where \( x_i \Rightarrow_L x_{i+1} \) (\( x_i \Rightarrow_R x_{i+1} \) for the left-most case) for \( i = 0, 1, \ldots, n-1 \).

The set \( \{ x \mid P(x) \} \) is the set of \( x \) for which \( P(x) \) holds. \( L(G) \), the language generated by \( G \), is the set \( \{ x \mid x \) is a word over \( T \}, S \Rightarrow^* x \} \). The set of a symbol \( Q \) is the set \( L_L(Q) = \{ a \mid a \) is from \( N \cup T, Q \Rightarrow a w \} \). \( A \) is a right neighbour of \( Q \) if \( S \Rightarrow Q B y \) and \( A \) is from \( L_L(B) \).

The grammar will be often given in the natural form by giving the set of productions and if necessary, the initial symbol. The symbols occurring on the left-hand sides of productions are the non-terminals, the other ones are the terminals. Nonterminals are usually denoted by capital Latin letters. The rules are usually numbered from zero, the number zero has the production having the initial symbol as its left hand side. The grammar is in the standard form if its zeroth rule has the form \( S \rightarrow \rightarrow L S' \) and the initial symbol \( S \) does not occur elsewhere. Sometimes we shall abbreviate 1. \( A \rightarrow w \), 2. \( A \rightarrow v \) into 1.2. \( A \rightarrow w | v \).

2. Parsing in compilers

The discovery that the results of the formal language theory (especially the parsing methods) can be used in compiler construction was probably the greatest step forward in the compiler methodology (see Gries [11]).

The methods of parsing become soon a thrilling topic of the study of its own right. Soon the two basic strategies of parsing, the top-