Reviews of Books and Papers in the Computer Field

DONALD L. EPLEY, Reviews Editor
D. W. FIFE, A. I. RUBIN, R. A. SHORT, H. S. STONE
Assistant Reviews Editors

Please address your comments and suggestions to the Reviews Editor: Donald L. Epley,
Department of Electrical Engineering, University of Iowa, Iowa City, Iowa 52240

A. COMPILERS


Since BNF (context-free) grammars have been found so useful in
describing the syntax of programming languages, a large number of
parsing algorithms for context-free grammars have been developed
for use in compilers and compiler-writing systems. A significant class
of these algorithms have the following properties:

1) The algorithm works only on a subset $S$ of the context-free
grams.

2) Given a grammar from $S$, a construction algorithm compiles a
parser which is tailored to that particular grammar.

3) The speed of the compiled parser is $C_n^2$, where $n$ is the length
of the string being parsed and $C_1$ is independent of the size of
the grammar or $n$.

4) The space required by the compiled parser is $C_n^2 + C_b$, where
$C_1$ is independent of $n$ or the grammar but $C_1$ depends on the
grammar.

The crucial parameters in evaluating an algorithm of this type
are $C_1$, $S$, and $C_2$. $C_3$ is usually negligible. If $S$ is too small, one
may have difficulty getting a grammar for the programming language
which the parser will accept. If $C_1$ is too large, the parser may be too
slow, and if $C_2$ grows too quickly with the size of the grammar, then
the compiler parser may not be used for medium or large grammars. The
algorithm described in this paper is of the type having these four
properties, so we will evaluate it by the above criteria.

Its main advantage seems to be a small $C_4$, making it quite fast.
This is obtained by using a transition matrix as a "switching matrix
which lets one determine from the top symbol of the stack (denoting
a row of the table) and the next symbol of the program to be pro-
cessed (represented by a column of the table) whether a reduction
should be made, or whether the incoming symbol should be pushed
onto the stack." This technique has been used in other compilers,
and this paper is an attempt to formalize and automate it.

The set of grammars which are accepted by the algorithm is a sub-
set of the operator grammars which satisfies certain complex restric-
tions. The author mentions that it is a subset of $(1, 1)$ bounded
context grammars, but gives no other indication of its relationship
to the sets of grammars acceptable to other fast parsers. Most gram-
mars for full programming languages are not operator grammars, and
many are not even $(1, 1)$ bounded context, so the set of acceptable
grammars for this algorithm is probably smaller than would be pref-
erable. This difficulty can be overcome by changing the grammar
(without changing the language it generates), but I suspect that it
is rather difficult and time consuming to maneuver a large grammar
into exactly the form necessary to satisfy the restrictions of this
algorithm.

The space $C_3$ is taken up mainly by the transition matrix. Its two
dimensions are the number of terminals in the grammar and the num-
ber of "starred" nonterminals added to the grammar (the construc-
tion algorithm first converts its operator grammar into a special
form, adding some nonterminals). This second number seems to be
about as large as the number of original nonterminals in the grammar.
The matrix size for an ALGOL compiler is $45 \times 100$, so the space used
is large, but not enough to exceed the memory of most large com-
puters, except possibly for extremely large grammars, such as PL/1.

In conclusion, the algorithm is quite fast, and if one has either a
large memory or a small grammar and is willing to spend the effort
needed to produce an acceptable grammar for his language, then the
algorithm could be quite useful.

I would have preferred to see more comparison of this algorithm
with other similar algorithms (such as Floyd's operator precedence
algorithm) by the three criteria I have mentioned. I found the mathe-
матическая часть of the paper complicated and hard to follow, but the
description of the implementation and the rest of the paper are clearly
written and the examples well chosen.

At the end of the paper, a section is devoted to the use of parsing
algorithms for operations other than the usual syntax checking.
While it has little to do with the rest of the paper, it is interesting.

Line (4.8) contains an error; it should read

$$U_1^* U_2^* \cdots U_{n-1}^* U_{n-1} U_1 T_1 \cdots T_n.$$  

JAY EARLEY
Dept. of Computer Science
University of California
Berkeley, Calif.

B. TIME-SHARED SYSTEM SCHEDULING

R68-47 Computer Scheduling Methods and Their Countermeasures
—E. G. Coffman, Jr., and L. Kleinrock (1968 Spring Joint Computer
pp. 11–21).

This narrative comprises a rather subjective and limited sum-
mmary of queuing analyses that are applicable to the design of com-
puter scheduling algorithms. The queuing disciplines are all treated