Introduction

LR(k) parsing was introduced by Knuth [17] in 1965. While the method was interesting and was sufficiently powerful to be able to parse any deterministic context-free language, there were serious obstacles to its use in practice. The method in [17] leads to table-driven parsers in which the size can grow exponentially with the number of productions. (An example may be found in [4].) In [18], Korenjak proposed to partition grammars to obtain smaller parsers but the method was somewhat ad hoc. Major progress was made by the introduction of SLR(k) grammars by deRemer [3]. By first constructing an LR(0) parser and then augmenting the LR(0) states with lookahead, deRemer got SLR(1) parsers with size independent of the number of instances in which lookahead was necessary. This led to smaller parsers than those obtained previously. The use of such a parser construction suggested that further space economies might be realized. Pager [19] found a minimal finite automaton from which to construct the parser. In [1], an approach is presented which attempts to merge states of the parsing automaton. Unfortunately, this procedure is not completely algorithmic.

Another approach is given in [15,16] in which minimal parsers are given for a special class of grammars.

In [6], a technique called characteristic parsing was introduced. This modification of LR parsing allows merging of states and produces very small parsers. The technique is general and so must be tuned to the characteristics of the family of grammars involved. For instance, the method is applied to the strict deterministic grammars [13,14] in [6].

*Research supported by the National Science Foundation under Grant GJ-474.

(1) If a parser is represented as a graph, the nodes of the graph correspond to what we will call "states." In the description of parsing automata, designations of these nodes become pushdown symbols and are therefore called tables by Aho and Ullman [2]. They correspond to the state sets of Knuth [17] or the configuration sets of deRemer [3]. Our measure of parser size is the number of such "states."
Simple precedence parsing [21] was introduced at approximately the same time as LR(k) parsing and much has been learned about precedence parsing subsequently [2,7,8,11,12,20]. The algorithms for constructing precedence parsers are relatively simple and the parsers often can be implemented to operate very rapidly.

In the present paper, we introduce a new type of parsing called production prefix parsing which is an outgrowth of characteristic parsing and of previous work on precedence parsing. The connections with the theory of characteristic parsing allow us to investigate the size of these parsers. The relationship with precedence parsing stems from the parser used for error recovery in [10]. Production prefix parsers exhibit the same sort of "locality" that is exploited in that work, suggesting that similar error recovery techniques may be applicable for these parsers.

The present extended abstract is meant to explain the ideas involved in an informal manner. A more formal presentation will appear in a sequel. We shall use standard terminology without repeating the definitions here. See [5,13] in which most of these terms are defined.

In production prefix parsing, the states of the parsing automaton correspond to the prefixes of right hand sides of productions in the grammar (the "production prefixes"). Thus, in effect, the state of the parser always records the portions of productions that have been parsed, obviating the need for look-ups in a production table. However, unlike SLR(k) parsers, the states of these parsers do not depend on context to the left of the current productions.

Production prefix parsers resemble some implementations of precedence parsers that have been used in the past [10]. However, although precedence information is implicit in the states, it is not necessary to refer to the precedence relations to do phrase detection. Additionally, we will show that our techniques can be used for a wider class of grammars.

We first give two examples of production prefix parsers. We then discuss the class of grammars for which production prefix parsing works, namely, the production prefix grammars. Next, we discuss production prefix parsers as characteristic parsers. Characteristic parsing then serves as a structure through which we can compare the size of a production prefix parser with the size of an SLR(1) parser for some production prefix grammar. We shall conclude the paper by considering the generality of production prefix grammars.

The Parsing Method

We first construct a parsing graph from the grammar to be parsed. The set of nodes on states of the graph are the set of distinct prefixes of right hand sides of the productions (the "production prefixes")\(^{(2)}\).

\(^{(2)}\)These are called "suitable prefixes" in [6].