Information Processing Letters 26 (1987/88) 251–253
North-Holland

11 January 1988

LOOPING LR PARSERS

Eljas SOISALON-SOININEN and Jorma TARHIO *

Department of Computer Science, University of Helsinki, Tukholmankatu 2, SF-00250 Helsinki, Finland

Communicated by A.V. Aho
Received 9 September 1986
Revised 15 April 1987

An LR parser generated from an arbitrary grammar may be looping. A characterization is given for grammars that will never produce a looping LR parser. A testing algorithm is outlined, and its time complexity is analyzed.

Keywords: Compiler, parsing

1. Introduction

Aho et al. [1] have introduced a method to construct deterministic LR parsers for some ambiguous grammars. A deterministic parser is achieved by modifying ambiguous entries in the parsing table generated. In the same way, it is possible to construct a deterministic parser from an arbitrary non-LR(k) grammar. However, the modified parser may accept only a subset of the original language.

To get a deterministic parser, one cannot arbitrarily disambiguate the ambiguous entries, because some choices may cause a looping parser. In the following, we shall characterize a 'loop-safe' grammar, i.e., a grammar whose parser will never get into a loop no matter how the conflicts are resolved.

The paper is organized as follows. Two examples of looping parsers are given in Section 2. A characterization of loop-safe grammars is presented in Section 3. A testing algorithm for loop-safeness is outlined and analyzed in Section 4.

We assume the reader to be familiar with the basic concepts of LR parsing (see, e.g., [2]).

* The work of this author was supported by the Academy of Finland.

2. Looping parsers

An LR parser is *looping* if there is a parsing configuration such that all subsequent parsing actions are reductions. In other words, after a certain point in parsing, none of the actions in parsing can be either shift, error, or accept.

There are two kinds of looping situations. In Table 1, we show the SLR(1) parsing table for grammar

$$\{(1)\ S' \to S,\ (2)\ S \to XSb,\ (3)\ S \to a,\ (4)\ X \to \varepsilon\}.$$

In states 0 and 2, there is a shift–reduce conflict between shift on "a" (s5) and reduction by $X \to \varepsilon$ (r4). If the alternative r4 is selected in both cases, the parser will be looping, because reduction by $X \to \varepsilon$ will be repeated until a stack overflow.

Table 1

State	action			goto	
	a	b	$	S	X
0	s5/r4			1	2
1			acc		
2	s5/r4			3	2
3		s4			
4		r2	r2		
5		r3	r3		

Table 2

State	action			goto		
	a	b	$	S	A	B
0	s5			1	2	4
1			acc			
2		s3/r5				
3			r2			
4		r3				
5		r4				

Let us consider another example. In Table 2, we show the SLR(1) parsing table for grammar

$$\{(1)\ S' \to S,\ (2)\ S \to Ab,\ (3)\ A \to B,\ (4)\ B \to a,\ (5)\ B \to A\}.$$

Again, we have a shift–reduce conflict. If the alternative r5 in state 2 is selected, the parser will repeat reductions by $A \to B$ and $B \to A$ forever. Note that there is no stack overflow in this case.

We carried out some tests with two parser generators: Yacc [3] and HLP84 [4]. In both systems, the user is [illegible] control the choices done by the generator in conflict situations, and so we managed to get the generated parsers into a loop in both cases described above.

3. Characterization of loop-safe grammars

Let $G = (N, T, P, S)$ be a context-free grammar.

Theorem. *G is a loop-safe grammar if and only if there is no* $A \in N$ *such that* $A \Rightarrow^+ \alpha A \beta$, *where* $\alpha \Rightarrow^* \varepsilon$ *and, whenever* $\alpha = \varepsilon$, $\beta \Rightarrow^* \varepsilon$.

Proof. If there is an $A \in N$ such that $A \Rightarrow^+ \alpha A \beta$, where $\alpha \Rightarrow^* \varepsilon$ and, whenever $\alpha = \varepsilon$, $\beta \Rightarrow^* \varepsilon$, the grammar is not LR and it is easy to modify the ambiguous entries to make the parser looping.

Conversely, let us assume that we have a looping LR parser constructed from grammar G. We consider the LR automaton for G. There are two cases:

(i) There is a path in the LR automaton such that it is a loop and its transition symbols $C_1, C_2, \ldots, C_n$ are nullable nonterminals. Then, G must produce $A \Rightarrow^* C_{1+i}C_{2+i} \ldots C_{n+i}A\beta$ for some $A \in N$ and for some i, $0 \leqslant i < n$, where $C_k = C_{k-n}$ for $k = n+1, n+2, \ldots, 2n$. By setting $\alpha = C_{1+i}C_{2+1} \ldots C_{n+i}$, we have that $A \Rightarrow^+ \alpha A \beta$, $\alpha \Rightarrow^* \varepsilon$, and $\alpha \neq \varepsilon$.

(ii) The LR automaton for G contains no such path described in case (i). Because the LR parser is looping, there is a threshold parsing configuration after which no shifts are performed. Because now all loops in the LR automaton contain at least one terminal or nonnullable nonterminal, no loop of the LR automaton can be followed in further parsing. So, there is only a finite number of parsing configurations that can happen, and there must be a parsing configuration $(\gamma A, w)$ that occurs several times, which implies $A \Rightarrow^+ A$. Thus, we have $A \Rightarrow^+ \alpha A \beta$, $\alpha \Rightarrow^* \varepsilon$, and $\beta \Rightarrow^* \varepsilon$. □

The characterization of loop-safeness immediately implies that no loops can occur if there are only shift–reduce conflicts and all of them are resolved in favour of shifting.

4. Testing

The characterization given in Section 3 can be applied in a preprocessing algorithm for parser generators allowing manipulation of parsing tables. Then, it becomes possible to warn the user of an input grammar which is not loop-safe.

Let R be a relation on N defined by: A R B, if $A \Rightarrow \alpha B \beta$ and $\alpha \Rightarrow^* \varepsilon$. Thus, $R^+(A)$, where R^+ denotes the transitive closure of R, contains all nonterminals that can start a sentential form derived from A. By computing the nullable nonterminals and the sets $R^+(A)$ for every $A \in N$, the test can easily be carried out if we use the following equivalent characterization:

> G is loop-safe if and only if there is no $A \in N$ such that $A \Rightarrow^+ A$ or $C \to B_1 B_2 \ldots B_n A \beta$ is a production where $C \in R^+(A)$ and $B_1, B_2, \ldots, B_n$, $n \geqslant 1$, are nullable nonterminals.

Let us estimate the testing time. The size of grammar G, denoted by $|G|$, is the sum of all $|A\omega|$, where $A \to \omega$ is a production in G. Since the relation R is of size $O(|G|)$, $R^+(A)$ can be computed in $O(|G|)$ time for every fixed A. Thus, all sets $R^+(A)$ can be computed in $O(|N|\,|G|)$ time and thus in $O(|G|^2)$ time.

Nullable nonterminals can be computed in $O(|G|)$ time, and testing the condition $A \Rightarrow^+ A$ can also be done in linear time. Thus, we conclude that the total time complexity is $O(|G|^2)$.

References

[1] A.V. Aho, S. Johnson and J.D. Ullman, Deterministic parsing of ambiguous grammars, Comm. ACM 18 (8) (1975) 441–452.

[2] A.V. Aho, R. Sethi and J.D. Ullman, Compilers: Principles, Techniques, and Tools (Addison-Wesley, Reading, MA, 1986).

[3] S. Johnson, Yacc—Yet Another Compiler Compiler, Computing Science Tech. Rept. 32, AT&T Bell Labs., 1975.

[4] K. Koskimies, O. Nurmi, J. Paakki and S. Sippu, The Design of the Language Processor Generator HLP84, Rept. A-1986-4, Dept. of Computing Science, Univ. of Helsinki, 1986.