Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Not Always a Full LR(1) Parser Generator

Roberto da Silva Bigonha

Relatório Técnico RT 022/91

Caixa Postal, 702
30.161 - Belo Horizonte - MG

Dezembro de 1991

Abstract

David Spector proposed an algorithm to efficiently generate compact LR(1) tables. However, his algorithm does not always generate full LR(1) parsing tables. This report identifies this problem and presents a proof that there are LR(1) grammars for which the Spector's parser generator fails to construct their LR(1) tables.

Keywords: LR grammars. LALR(1) grammars. LR parsing

Contents

1 Introduction

LR(1) parsers are more powerfull than LALR(1) parsers, but the latter have been considered the method of choice because LALR(1) tables are generally smaller than full LR(1) tables, and yet most programming languages features can be expressed by LALR(1) grammars.

Spector discusses this issue [5], and based on the fact that the minimal finite pushdown automatum produced by an full LR(1) parser generator is identical to that produced by an LALR(1) generator for an LALR(1) language [8], proposes an efficient algorithm to generate LR(1) tables, whose sizes are smaller than those produced by common LR(1) algorithm found in the literature [1, 2, 3, 7]. With the new method, Spector tries to establish the basis for full LR(1) parser to become the standard rather than the exception.

This report analyzes the proposed algorithm, proves that there are LR(1) grammars for which the Spector's parser generator fails to construct their LR(1) tables.

2 Terminology

It is assumed that the reader is familiar with the terminology and conventions concerning grammars and parsers [1, 2, 3]. Thus, just little of the usual terminology and conventions are repeated in the following.

Context-free grammars have the form $G = (N, \Sigma, P, S')$, where N is a finite set of non-terminal symbols, Σ is a finite set of terminal symbols, P is a finite set of productions, which are elements of (N, V^*), where $V = N \cup \Sigma$, and S' is the start symbol.

All grammars are assumed to be reduced and to have the production $S' \rightarrow S$, where S' does not occur in any other production.

Lowercase Greek letter, such as α, β, γ are in V^*; the Greek letter ϵ denotes the empty string; lowercase italic letters near the beginning of the alphabet, such as a, b, c, are in Σ; uppercase italic letters near the beginning of the alphabet, such as A, B, C, are in the set N of non-terminal symbols. The symbol $\vdash$ represents the end-of-file mark.

An LR(0) item is written in the form $[A \rightarrow \alpha \cdot \beta]$, where $A \rightarrow \alpha\beta$ is in P. Similarly, an LR(1) or LALR(1) item is written in the form $[A \rightarrow \alpha \cdot \beta, u]$, where $u \in \Sigma$. A canonical collection is a collection of sets of itens, each set corresponds to a parser state.

The following definitions have been adapted from [4].

Definition 1 *Let G be a context-free grammar. The LR(k) machine for grammar G is $LRM_k^G = (M_k^G, IS_k^G, GOTO_k^G)$, where $k \geq 0$, M_k^G is a set of LR(k) states, each state corresponds to a set of itens in the canonical collection of LR(k) itens. IS_k^G is the initial state. $GOTO_k^G$ is the state transition mapping of type: $M_k^G \times V \rightarrow M_k^G$.*

Definition 2 *Let I_i be a state of an LRM_1^G machine . Then*

$$CORE(I_i) = \bigcup\{[A \rightarrow \alpha \cdot \beta] \mid [A \rightarrow \alpha \cdot \beta, u] \in I_i\}$$

Definition 3 *Let $Q, T \imath M_0^G, X \in V$ and $\alpha \in V^*$. Then*

$$PRED(T, \alpha) = \begin{cases} \{T\} & \text{if } \alpha = \epsilon \\ \bigcup\{PRED(Q, \alpha') \mid GOTO_1^G(Q, X) = T\} & \text{if } \alpha = \alpha' X \end{cases}$$

The following theorem provides the basis to construct an algorithm to compute look-ahead sets of LR(1) itens by directly searching the LR(0) machine associated with the canonical collection.

Theorem 1 *Let $T, Q \in M_0^G$. Then*

1. $LR([S' \rightarrow \cdot S], IS_0^G) = \vdash$
2. $LR([A \rightarrow \alpha \cdot \beta], T) = \bigcup\{LR([A \rightarrow \cdot\alpha\beta], Q) \mid Q \in PRED(T, \alpha)\}$
3. $LR([A \rightarrow \cdot\alpha\beta], T) = \bigcup\{FIRST(\psi) \mid [B \rightarrow \varphi \cdot A\psi] \in T\} - \{\epsilon\} \quad \cup$
   $\bigcup\{LR([B \rightarrow \varphi \cdot A\psi], T) \mid [B \rightarrow \varphi \cdot A\psi] \in T \wedge \psi \stackrel{*}{\Rightarrow} \epsilon\}$

Proof: The proof of this theorem can be found in [4], and thus, is omitted here.

3 Spector's Parser Generator

Spector proposes to compute the full LR(1) look-ahead sets in three stages [5]:

1. Create the LR(0) collection and construct the LR(0) machine, as it is usually done in the construction of LALR(1) tables.
2. Apply an algorithm derived from Theorem 1 to compute the look-ahead sets of all itens, thus transforming the LR(0) machine into an LALR(1) machine [4].

3. Resolve conflicts of inadequate states by searching the LALR(1) machine in the following way:

   (a) Each inadequate state is split into separate states, one for each transition into it.

   (b) Then the look-ahead sets of the itens of each of these separate states are computed.

Spector successfully hand-applied his method to a variety of small grammars of various types and invited the reader to complete the algorithm and prove it correct, that is, that it generates full LR(1) parse tables for LR(1) grammars.

However, the proposed algorithm does not work correctly at all times. There are LR(1) grammar for which the above algorithm fails to produce their unambigous LR(1) table. In the following sections a counter-example is presented and a sub-class of these LR(1) grammars is identified.

4 An Counter-Example

Consider the following LR(1) grammar $G_1 = (N, \Sigma, P, S')$, where $N = \{S', S, A, B, D, E\}$, $\Sigma = \{a, b, c, d\}$ and the set of productions P is:

1.	S'	$\rightarrow$	S	6.	D	$\rightarrow$	Bc
2.	S	$\rightarrow$	aEc	7.	D	$\rightarrow$	A
3.	S	$\rightarrow$	bDd	8.	E	$\rightarrow$	A
4.	A	$\rightarrow$	ab	9.	E	$\rightarrow$	Bd
5.	B	$\rightarrow$	ab				

The canonical LR(1) colection for grammar G_1 computed via Spector's algorithm [5] is:

$$
\begin{array}{lll}
I_0: & S' \to \cdot S, \vdash \\
 & S \to \cdot aEc, \vdash \\
 & S \to \cdot bDd, \vdash \\
 & \\
I_1: & S' \to S\cdot, \vdash \\
 & \\
I_2: & S \to a \cdot Ec, \vdash \\
 & E \to \cdot Bd, c \\
 & E \to \cdot A, c \\
 & B \to \cdot ab, d \\
 & A \to \cdot ab, c \\
 & \\
I_3: & S \to b \cdot Dd, \vdash \\
 & D \to \cdot Bc, d \\
 & D \to \cdot A, d \\
 & B \to \cdot ab, c \\
 & A \to \cdot ab, d \\
 & \\
I_4: & B \to a \cdot b, c/d \\
 & A \to a \cdot b, c/d \\
 & \\
I_5: & B \to ab\cdot, c/d \\
 & A \to ab\cdot, c/d
\end{array}
\qquad
\begin{array}{lll}
I_6: & S \to aE \cdot c, \vdash \\
 & \\
I_7: & S \to aEc\cdot, \vdash \\
 & \\
I_8: & E \to B \cdot d, c \\
 & \\
I_9: & E \to Bd\cdot, c \\
 & \\
I_{10}: & E \to A\cdot, c \\
 & \\
I_{11}: & S \to bD \cdot d, \vdash \\
 & \\
I_{12}: & S \to bDd\cdot, \vdash \\
 & \\
I_{13}: & D \to A\cdot, d \\
 & \\
I_{14}: & D \to B \cdot c, d \\
 & \\
I_{15}: & D \to Bc\cdot, d
\end{array}
$$

State that corresponds to I_5 is inadequate because it presents a reduce-reduce conflict. Thus, according to Spector's paper grammar G_1 is incorrectly considered non-LR(1).

However, grammar G_1 is LR(1), although not LALR(1). In fact, the following LR(1) canonical collection with no conflicts was computed by algorithm proposed in [3]:

$$
\begin{array}{lll}
I_0: & S' \to \cdot S, \vdash \\
 & S \to \cdot aEc, \vdash \\
 & S \to \cdot bDc, \vdash \\
I_1: & S' \to S, \vdash \\
I_2: & S \to a \cdot Ec, \vdash \\
 & E \to \cdot Bd, c \\
 & E \to \cdot A, c \\
 & B \to \cdot ab, d \\
 & A \to \cdot ab, c \\
I_3: & S \to b \cdot Dd, \vdash \\
 & D \to \cdot Bc, d \\
 & D \to \cdot A, d \\
 & B \to \cdot ab, c \\
 & A \to \cdot ab, d \\
I_4: & B \to a \cdot b, d \\
 & A \to a \cdot b, c \\
I_5: & B \to a \cdot b, c \\
 & A \to a \cdot b, d \\
I_6: & B \to ab \cdot, d \\
 & A \to ab \cdot, c \\
I_7: & B \to ab \cdot, c \\
 & A \to ab \cdot, d \\
I_8: & S \to aE \cdot c, \vdash \\
I_9: & S \to aEc, \vdash \\
I_{10}: & E \to B \cdot d, c \\
I_{11}: & E \to Bd \cdot, c \\
I_{12}: & E \to A \cdot, c \\
I_{13}: & S \to bD \cdot d, \vdash \\
I_{14}: & S \to bDd \cdot, \vdash \\
I_{15}: & D \to A \cdot, d \\
I_{16}: & D \to B \cdot f, d \\
I_{17}: & D \to Bf \cdot, d
\end{array}
$$

Therefore, there are LR(1) grammars for which Spector's method for computing full LR(1) parsing table fails. In fact, there are grammars for which the proposed mechanism of splitting inadequate states is not enough to recover all the information needed to resolve conflicts introduced when LR(1) states with the same core were merged. In the following section this class of grammar is characterized.

5 Generalization

Consider a grammar $G = (N, \Sigma, P, S')$, whose LR(1) canonical collection C_A has the form

$$C_A = \{\ldots, I_i, I_k, I_p, I_q, \cdots\}$$

where

- $I_i = \{[A \to \alpha \cdot b, d], [B \to \alpha \cdot b, c]\}$
- $I_k = \{[A \to \alpha b \cdot, d], [B \to \alpha b \cdot, c]\}$
- $I_p = \{[A \to \alpha \cdot b, c], [B \to \alpha \cdot b, d]\}$
- $I_q = \{[A \to \alpha b \cdot, c], [B \to \alpha b \cdot, d]\}$
- $\text{GOTO}_1^G(I_i, b) = I_k$
- $\text{GOTO}_1^G(I_p, b) = I_q$

Notice that $CORE(I_i) = CORE(I_p)$ and $CORE(I_k) = CORE(I_q)$.

Assume that $CORE(I_i) \cap CORE(I_j) = \Phi$, for all $j \neq p$ and $j \neq i$, and that, for all $s \neq q$ and $s \neq k$, $CORE(I_k) \cap CORE(I_s) = \Phi$. These assumptions imply that I_k and I_q have each just one predecessor state, which are I_i and I_q, respectively.

The lookahead sets for states I_k and I_q are disjoint, therefore these states are not inadequate. Assuming that C_A does not have any other inadequate states, grammar G is LR(1).

On the other hand, if Spector's algorithm had been used to compute LR(1) itens, the resulting canonical collection C_S would have the states I_r and I_t in place of I_i, I_k, I_p and I_q, such that:

- $I_r = \{[A \to \alpha \cdot b, c/d], [B \to \alpha \cdot b, c/d]\}$
- $I_t = \{[A \to \alpha b \cdot, c/d], [B \to \alpha b \cdot, c/d]\}$
- $\text{GOTO}_1^G(I_r, b) = I_t$

State I_t of C_S is clearly inadequate. Since I_r is the only predecessor state of I_t, it is not possible to split I_t into separate states, and, therefore, the conflict remains unresolved, although G had been assumed to be an LR(1) grammar.

6 Conclusions

There is a class of LR(1) grammars for which the algorithm proposed in [5] fails to produce full LR(1) parser tables. Grammars in this class have associated LR(0) canonical collection containing inadequate states that cannot be separated because they all have only a single transition into them. This usually happens when two or more path containing more than

one state in the LR(1) machine are merged into a common path in the LR(0) machine. With this class of grammars, it would be necessary to modify the splitting algorithm to backwardly search the LR(0) machine for the appropriate predecessors of the inadequate state that can be split.

References

[1] Aho , A. V.,and Ullman, J. D., *The Theory of Parsing, Translation, and Compiling*, Vols. 1 e 2, Prentice-Hall, Englewood Cliffs, N.J., 1972.

[2] Aho, A. V. and Johnson, *Principles of Compiler Design*, Addison-Wesley Publishing Company Co, 1977.

[3] Aho A. V., Sethi, R. and Johnson, *Principles of Compiler Design*, Addison-Wesley Publishing Company Co, 1986.

[4] Kristensen, B. B. and Madsen, O. L., "Methods for Computing LALR(k) Lookahead", ACM Transaction on Programming Languages and Systems, Vol 3, Nº 1, January 1981, pp 60–82.

[5] Spector, David, "*Full LR(1) Parser Generation*", ACM Sigplan Notices, Vol. 16, Nº 8, August 1981.

[6] Spector, David, "*Efficient Full LR(1) Parser Generation*", ACM Sigplan Notices, Vol. 23, Nº 12, December 1988.

[7] Pager, David, "*A Practical General Method for Constructing LR(k) Parsers*", Acta Informatica 7, 249-268, 1977.

[8] Wethrell, C.S, and Shannon, A.S., Letter to the Editor, ACM SIGPLAN Notices, Volume 14, Nº 3, March 1979, pp 3.