ISSN 0105-8517

METHODS FOR LR(k) TESTING

(Informative Diagnostics on LALR(k)-Conflicts)

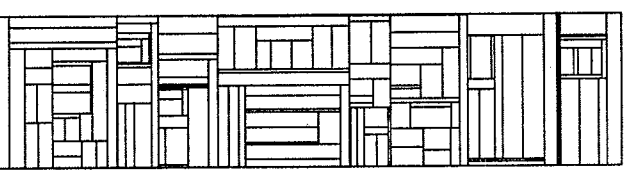
by

Bent Bruun Kristensen*
and
Ole Lehrmann Madsen

DAIMI PB-106
November 1979

* Aalborg University Center, Aalborg, Denmark.

Computer Science Department
AARHUS UNIVERSITY
Ny Munkegade – DK 8000 Aarhus C – DENMARK
Telephone: 06 – 12 83 55



Abstract

Methods for LR(k) testing based on the LR(0)-machine are discussed. Theoretical results on LR(k)-conditions are proved, and an algorithm for LR(k) testing derived from these results is given. The application of the method as a means of giving informative diagnostics on LALR(k)-conflicts is presented. Finally various modifications of the method are suggested to improve the efficiency of the LR(k) testing algorithm.

1. INTRODUCTION

A practical general method for LR(k) testing is presented. The approach assumes the existence of the LR(0)-machine, and the computation is based directly on this machine. The method may be used in connection with an SLR(k) or LALR(k) parser generator system, in order to supply the user with informative diagnostics when a grammar fails to obey one of the above conditions.

We think that the LALR(k)-grammars are a sufficient class for most applications. However the LALR(k) concept seems to be more difficult to comprehend than the LR(k)-concept. The information which may be a product of our LR(k) test is an understandable and operational foundation for modifications of non-LALR(k) - or SLR(k)-grammars. .

When an LALR(k)-conflict is detected between two items I, J in a state T of the LR(0)-machine, the information on the conflict may, as a minimum, be a description of T, - especially I and J -, and the two conflicting LALR(k) lookaheadsets for I and J in T. A better diagnostic is for each of the items I and J to describe the items and the states of the predecessor tree, which contribute to the LALR(k) lookahead, and the actual contributions as well. Alternatively we may, based on the pair (I,J) in T, find the predecessor paths in terms of pairs of items (I', J') in states T' leading to (I,J) in T. By computing at each step on a path the lookahead, which (I', J') in T' contributes with to its successor, we obtain a pair of lookaheadsets for (I,J) in T corresponding to each of the possible predecessor paths of this kind. These sets are essentially LR(k)-lookaheadsets for (I,J) in the copies of T in the LR(k)-machine. Thus we may actually perform LR(k) tests for (I,J) in "T". More important is that the information given by these predecessor paths, the (I', J') in T', and the contributions from (I', J') in T', forms relevant and sufficient diagnostics on the LALR(k)-conflict between I and J in T.

A number of approaches have been given in the area of LR(k)-testing: The original LR(k) construction method given in [Knuth65] requires an excessive amount of time and space to produce parsers for practical

grammars. A number of more efficient variations of this method have been presented for various subsets of the LR(k) grammars. Among these are the SLR(k)- and LALR(k)-methods by [DeRemer69, 71], where the idea is to employ an LR(0) algorithm even for non-LR(0) grammars and then to extend the LR(0) parser by various kinds of lookahead.

In [DeRemer69] a theoretical method is outlined to extend the LR(0)-machine to cover LR(k)-grammars by state-splitting. [Pager77a] contains a practical method based on this idea.

Another way of optimizing LR(k)-parser construction is to combine the states of an LR(k)-parser as they are generated, reducing the number of configurations to be evaluated as well as the space used for the evaluation. In [Pager77b] two criteria are given to be applied in this scheme. In addition to LR(k)-parser construction, methods have been introduced for LR(k)-testing only. [Hunt et al. 75] gives a method for LR(k)-testing with a low complexity. The method includes no automatic LR(k)-parser construction.

The rest of the present paper is organized as follows:
Chapter 2 is a summary of terminology and important basic results.
An informal outline of the method for LR(k)-testing is given in Chapter 3, which is concluded by formalizing the results used in the method.
Chapter 4 contains an actual algorithm for LR(k)-testing and a proof of its correctness. In Chapter 5 the use of the method to produce informative diagnostics is discussed. Furthermore various improvements and variants of the method are described. Appendix A shows the use of the algorithm to test an example grammar.

Appendix B contains algorithms realizing the variants and improvements described in Chapter 5.

Upper bounds for the complexity of the presented algorithms are given in the respective sections.

Acknowledgment

During the preparation of this paper we have received many helpful comments from Peter Kornerup.

2. BASIC TERMINOLOGY AND RESULTS

The reader is assumed to be familiar with the terminology and conventions from [Aho & Ullman 72] concerning grammars and parsers. Especially the following concepts are used extensively: $FIRST_k$, EFF_k, $\oplus_k$, LR-item, (canonical) collection of sets of LR(k)-items, GOTO, CORE, e (the empty string), and viable prefix.

A context-free grammar is always assumed to have the form G = (N, Σ, P, S) where N is a finite set of nonterminal symbols, Σ is a finite set of terminal symbols, P is a finite set of productions, and S is the start symbol. All grammars are assumed to be free of "useless" symbols. They are also assumed to be extended with a new start symbol S' and the production $S' \rightarrow S \dashv^k$, where $\dashv$ is a symbol not in $(N \cup \Sigma)$.

We use the following conventions: small Greek letters such as α, β, γ are in $(N \cup \Sigma)^*$; small Latin letters in the beginning of the alphabet such as a, b, c are in Σ; small Latin letters in the end of the alphabet such as v, x, y are in Σ^*; capital Latin letters in the beginning of the alphabet such as A, B, C are in N; capital Latin letters in the end of the alphabet such as X, Y, Z are in $(N \cup \Sigma)$.

If M is a set of subsets of some set N then $\cup M$ means

$$\{x \in N \mid x \in m \ \& \ m \in M\}.$$

We shall repeat some definitions and theorems often in a slightly modified form.

Definition 2.1

Let G be a CFG, then the LR(k)-machine for G is

$LRM_k^G = (M_k^G, IS_k^G, GOTO_k^G)$, where

M_k^G : is a set of (LR(k)-)states, one for each set of items in the canonical collection of LR(k)-items. We do not distinguish between a state and its corresponding set of items.

IS_k^G : is the initial state.

$GOTO_k^G$: is the GOTO-function defined on $M_k^G \times (N \cup \Sigma) \to M_k^G$ □

For a given grammar G we will assume the existence of its LRM_k^G on this form. The superscript G is omitted when this causes no confusion. $GOTO_k$ is extended in the obvious way to $(M_k) \times (N \cup \Sigma)^* \to M_k$.

The following definitions and theorems summarize the notions and results necessary in the following sections:

Definitions

Let G be a CFG, with LR(k)-states M_k, $k \geq 0$.

(2.2) Let $T \in M_k$, then

$$LR_k([A \to \alpha.\beta], T) = \{u \mid [A \to \alpha.\beta, u] \in T\}.$$

(2.3) Let $[A \to \alpha.\beta, u]$ be a LR(k)-item and let $S \in M_k$, then
$CORE([A \to \alpha.\beta, u]) = [A \to \alpha.\beta]$, and
$CORE(S) = \{CORE(I) \mid I \in S\}$.
We shall not distinguish between the items $[A \to \alpha.\beta, e]$ and $[A \to \alpha.\beta]$.

(2.4) Let $T \in M_0$, then

$$URCORE_k(T) = \{S \in M_k \mid CORE(S) = T\}.$$

(2.5) $$LALR_k([A \to \alpha.\beta], T) = \cup\{LR_k([A \to \alpha.\beta], S) \mid S \in URCORE_k(T)\}.$$

(2.6) G is said to be LR(k), $k \geq 0$, if for all $T \in M_k$ and for all distinct items $[A \to \alpha.\beta, u]$ and $[B \to \gamma., v]$ in T, we have

$$v \notin FIRST_k(EFF_k(\beta)\, u),$$

or equivalently that

(*) $$EFF_k(\beta) \oplus_k LR_k([A \to \alpha.\beta], T) \cap LR_k([B \to \gamma.], T) = \emptyset.$$

(*) The $\oplus_k$-operator has higher precedence than the $\cap$-operator.

(We shall use 2.6 as a definition. It may be found in [Aho & Ullman 72] as a theorem.)

(2.7) G is said to be LALR(k), $k \geq 0$, if for all $T \in M_0$, and for all distinct items $[A \rightarrow \alpha.\beta]$ and $[B \rightarrow \gamma.]$ in T we have

$$EFF_k(\beta) \oplus_k LALR_k([A \rightarrow \alpha.\beta], T) \cap LALR_k([B \rightarrow \gamma.], T) = \emptyset.$$

(2.8) Let $T \in M_k$, $X \in (N \cup \Sigma)$ and $\alpha \in (N \cup \Sigma)^*$, then

$$PRED(T,\alpha) = \begin{cases} \{T\} & \text{if } \alpha = e \\ \cup\{PRED(S,\alpha') \mid GOTO_k(S,X) = T\} & \text{if } \alpha = \alpha'X \end{cases}$$

□

Theorems

(2.9) Let $[A \rightarrow \alpha.\beta] \neq [S' \rightarrow .S \dashv^k]$ then

$$LALR_k([A \rightarrow \alpha.\beta], T) =$$
$$\cup\{FIRST_k(\Psi) \oplus_k LALR_k([B \rightarrow \varphi.A\Psi], S) \mid$$
$$S \in PRED(T,\alpha) \wedge [B \rightarrow \varphi.A\Psi] \in S\}$$

(2.10) $LALR_1([A \rightarrow \alpha.\beta], T) = \cup\{L(S,A) \mid S \in PRED(T,\alpha)\}$

where

$$L(S,A) = \cup\{FIRST_1(\Psi) \mid [B \rightarrow \varphi.A\Psi] \in S\} \setminus \{e\}$$
$$\cup \cup\{LALR_1([B \rightarrow \varphi.A\Psi], S) \mid [B \rightarrow \varphi.A\Psi] \in S \wedge \Psi \Rightarrow^* e\}$$

(2.11) $\forall S \in PRED(T,\alpha) : LR_k([A \rightarrow \alpha.\beta], T) = LR_k([A \rightarrow .\alpha\beta], S)$

(2.12) Let $[A \rightarrow .\alpha] \neq [S' \rightarrow .S \dashv^k]$ then

$$LR_k([A \rightarrow .\alpha], T) =$$
$$\cup\{FIRST_k(\Psi) \oplus_k LR_k([B \rightarrow \varphi.A\Psi], T) \mid [B \rightarrow \varphi.A\Psi] \in T\}.$$

(2.13) $LR_k([A \rightarrow \alpha.\beta], T) = \{w \mid w \in FIRST_k(y) \wedge$
$S' \Rightarrow_{rm}^* \gamma Ay = \gamma\alpha\beta y \wedge GOTO_k(IS_k, \gamma\alpha) = T\}$.

Proofs

2.9 and 2.10 have been proved in [Kristensen & Madsen 79a]. 2.11, 2.12 and 2.13 follow directly from section 5.2.3 in [Aho & Ullman 72].

□

3. PROPERTIES OF LR-CONDITIONS

Conflicts in the LR(0)-machine which cannot be solved by supplying LALR(k)-lookeahead, may disappear by considering the LR(k)-machine instead. LR(k)-lookahead cannot just be added to the LR(0)-machine but we may either construct the LR(k)-machine - or split certain states of the LR(0)-machine into a number of copies. Given a conflict between two items $[A \to \alpha.\pi]$ and $[B \to \delta.]$ in a state T in M_0, do then any conflicts exist between these items in any state of M_k with CORE identical to T?

Using definitions 2.2 and 2.4 this may be expressed as the condition:

$$[3.1] \qquad EFF_k(\pi) \oplus_k LR_k([A \to \alpha.\pi], S) \cap LR_k([B \to \delta.], S) = \emptyset$$

for all $S \in URCORE_k(T)$.

[3.1] may be reformulated based on the predecessor states of $S \in URCORE_k(T)$, using theorems 2.11, 2.12, as the following equivalent condition:

<u>case $\alpha = \alpha'\delta$:</u>

$$[3.2] \qquad EFF_k(\pi) \oplus_k LR_k([A \to \alpha'.\delta\pi], R) \cap FIRST_k(\Psi) \oplus_k LR_k([C \to \varphi.B\Psi], R) = \emptyset$$

for all $[C \to \varphi.B\Psi] \in R$ for all $R \in PRED(S, \delta)$, and

<u>case $\delta = \delta'\alpha$:</u>

$$[3.3] \qquad EFF_k(\pi) \oplus_k FIRST_k(\Psi) \oplus_k LR_k([C \to \varphi.A\Psi], R) \cap LR_k([B \to \delta'.\alpha], R) = \emptyset$$

for all $[C \to \varphi.A\Psi] \in R$ for all $R \in PRED(S, \alpha)$. Either of [3.2] and [3.3] may in turn be expressed by a new set of predecessor states, etc.

It may be seen that we have achieved a recursive formulation. At each step an additional set of strings may be concatenated to the sets of strings produced already. The recursion may be stopped as soon as each string produced has a length greater than or equal to k, in which case the two sets of strings may be compared directly.

The predicate $LRCOND_k$ will express the condition discussed above based on the LR(0)-machine for two sets of strings, $\mathfrak{A}, \mathfrak{B} \subseteq \Sigma^{*k}$, for two items $[A \to \alpha.\pi]$, $[B \to \beta.\delta] \in T$.

Definition 3.4

Let $T \in M_0$ and $[A \to \alpha.\pi]$, $[B \to \beta.\delta] \in T$ and $\mathfrak{A}, \mathfrak{B} \subseteq \Sigma^{*k}$, then

$$LRCOND_k(\mathfrak{A}, [A \to \alpha.\pi], \mathfrak{B}, [B \to \beta.\delta], T) \equiv$$

$$\forall\ S \in URCORE_k(T):$$

$$\mathfrak{A} \oplus_k LR_k([A \to \alpha.\pi], S) \cap \mathfrak{B} \oplus_k LR_k([B \to \beta.\delta], S) = \emptyset$$

□

The LR(k)-condition for the grammar G may then be expressed, based on its LR(0)-machine (LRM_0) using $LRCOND_k$, as an immediate consequence of definition 2.6 and 3.4:

Theorem 3.5

G is LR(k) if and only if

$\forall\ T \in M_0$:

$\forall\ [A \to \alpha.\pi], [B \to \beta.] \in T$:

$$LRCOND_k(EFF_k(\pi), [A \to \alpha.\pi], \{e\}, [B \to \beta.], T)$$

□

Returning to our starting point we wanted to perform the computation of the LR-condition on the LR(0)-machine rather than on the LR(k)-machine. As outlined above $LRCOND_k$ for some sets of strings and some items in a state T may be expressed by $LRCOND_k$ for some parameter sets produced by the PRED-operator on the LR(k)-machine, LRM_k. Thus we have an operation-sequence:

$$URCORE_k, PRED, \ldots PRED.$$

We claim that this sequence is equivalent to the following, using the PRED-operator on the LR(0)-machine, M_0:

$$PRED, \ldots, PRED, URCORE_k$$

This is a recursive formulation based on LRM_0, except for the last operation, $URCORE_k$. Below we discuss different stop-criterias which will enable us to eliminate the $URCORE_k$-operator, thus giving a formulation based entirely on LRM_0.

Let $G \subseteq \Sigma^{*k}$ be the set of strings produced before the contribution from the item $[A \rightarrow \alpha.\beta]$ in state T is concatenated. One criterion is obvious, namely

(*) $$|G|_{min} \geq k.$$

Another criterion is possible when $\alpha = e$. In this case the state T may itself contribute with a set of strings, C, for the item $[A \rightarrow .\beta]$, such that

$$|G \oplus_k C|_{min} \geq k.$$

These criteria may be expressed formally by the predicate $BOTTOM_k$.

Definition 3.6

Let $T \in M_0$ and $[A \rightarrow \alpha.\beta] \in T$ and $G \subseteq \Sigma^{*k}$, then

$$BOTTOM_k(G, [A \rightarrow \alpha.\beta], T) \equiv$$
$$(\alpha \neq e \Rightarrow |G|_{min} \geq k) \wedge$$
$$(\alpha = e \Rightarrow$$
$$\forall\ [B \rightarrow \varphi.A\Psi] \in T:$$
$$BOTTOM_k(G \oplus_k FIRST_k(\Psi), [B \rightarrow \varphi.A\Psi], T))$$

□

An effect of $BOTTOM_k$ is that we may restrict ourselves to state T and need not consider $URCORE_k(T)$.

(*) Let $G \subseteq \Sigma^{*k}$ then $|G|_{min}$ is the length of the shortest string in G, if $G \neq \emptyset$, - and 0 if $G = \emptyset$.

Lemma 3.7

Let $T \in M_0$ and $[A \to \alpha.\beta] \in T$ and $\mathfrak{A} \subseteq \Sigma^{*k}$.
If $BOTTOM_k(\mathfrak{A}, [A \to \alpha.\beta], T)$ then

$$\forall S \in URCORE_k(T):$$

$$\mathfrak{A} \oplus_k LR_k([A \to \alpha.\beta], S) = \mathfrak{A} \oplus_k LALR_k([A \to \alpha.\beta], T)$$

Proof: Definitions 2.5 and 3.6 and theorem 2.11.

□

Theorem 3.8

Let $T \in M_0$ and $[A \to \alpha.\pi]$, $[B \to \beta.\delta] \in T$ and $\mathfrak{A},\mathfrak{B} \subseteq \Sigma^{*k}$.
If $BOTTOM_k(\mathfrak{B}, [B \to \beta.\delta], T)$ then

$$LRCOND_k(\mathfrak{A}, [A \to \alpha.\pi], \mathfrak{B}, [B \to \beta.\delta], T) \equiv$$

$$\mathfrak{A} \oplus_k LALR_k([A \to \alpha.\pi], T) \cap \mathfrak{B} \oplus_k LALR_k([B \to \beta.\delta], T) = \emptyset$$

Proof: Lemma 3.7 and definition 2.5.

□

A problem exists when we try to transform the above recursive formulation of the LR-condition into an algorithm. In general the LR(0)-machine contains a number of cycles which may cause termination problems. Before we solve this problem and introduce an algorithm, we shall restate the above discussion on the recursive formulation of $LRCOND_k$ (i.e. [3.1], [3.2], [3.3]) in formal terms.

$LRCOND_k$ with items, sets and a state as arguments may be characterized by the set of predecessor states in LRM_0.

Lemma 3.9

Let $T \in M_0$ and $[A \to \alpha\beta.\pi]$, $[B \to \beta.\delta] \in T$ and $\mathfrak{A},\mathfrak{B} \subseteq \Sigma^{*k}$, then

$$LRCOND_k(\mathfrak{A}, [A \to \alpha\beta.\pi], \mathfrak{B}, [B \to \beta.\delta], T) \equiv$$

$$\forall S \in PRED(T, \beta):$$

$$LRCOND_k(\mathfrak{A}, [A \to \alpha.\beta\pi], \mathfrak{B}, [B \to .\beta\delta], S)$$

Proof: Definition 3.4 and theorem 2.11 and the fact that

$$CORE(GOTO_k(R, X)) = GOTO_0(CORE(R), X).$$

□

Lemma 3.10

Let $T \in M_0$ and $[A \to \alpha.\pi]$, $[B \to .\delta] \in T$ and $\mathfrak{A}, \mathfrak{B} \subseteq \Sigma^{*k}$, then

$$LRCOND_k(\mathfrak{A}, [A \to \alpha.\pi], \mathfrak{B}, [B \to .\delta], T) \equiv$$
$$\forall [C \to \varphi.B\Psi] \in T:$$
$$LRCOND_k(\mathfrak{A}, [A \to \alpha.\pi], \mathfrak{B} \oplus_k FIRST_k(\Psi), [C \to \varphi.B\Psi], T)$$

Proof: Using definition 3.4 and theorem 2.12 we have that

$$LRCOND_k(\mathfrak{A}, [A \to \alpha.\pi], \mathfrak{B}, [B \to .\delta], T) \equiv$$
$$\forall S \in URCORE_k(T):$$
$$\mathfrak{A} \oplus_k LR_k([A \to \alpha.\pi], S) \cap$$
$$\mathfrak{B} \oplus_k \cup\{FIRST_k(\Psi) \oplus_k LR_k([C \to \varphi.B\Psi], S) \mid [C \to \varphi.B\Psi] \in T\} = \emptyset$$
$$\equiv$$
$$\forall [C \to \varphi.B\Psi] \in T:$$
$$\forall S \in URCORE_k(T):$$
$$\mathfrak{A} \oplus_k LR_k([A \to \alpha.\pi], S) \cap$$
$$\mathfrak{B} \oplus_k FIRST_k(\Psi) \oplus_k LR_k([C \to \varphi.B\Psi], S) = \emptyset$$

The lemma then follows using definition 3.4.

□

The following theorem is an immediate consequence of combining lemma 3.9 and 3.10.

Theorem 3.11

Let $T \in M_0$ and $[A \to \alpha\beta.\pi]$, $[B \to \beta.\delta] \in T$ and $\mathfrak{A},\mathfrak{B} \subseteq \Sigma^{*k}$, then

$$LRCOND_k(\mathfrak{A}, [A \to \alpha\beta.\pi], \mathfrak{B}, [B \to \beta.\delta], T) \equiv$$

$$\forall S \in PRED(T, \beta):$$

$$\forall [C \to \varphi.B\Psi] \in S:$$

$$LRCOND_k(\mathfrak{A}, [A \to \alpha.\beta\pi], \mathfrak{B} \oplus_k FIRST_k(\Psi), [C \to \varphi.B\Psi], S)$$

□

The theorems 3.8 and 3.11 directly offer a solution for an algorithm for LR(k)-testing working only on the LR(0)-machine.

The theorems and lemmas in this section are apparently not symmetrical in the arguments of $LRCOND_k$. We may repair the defect by observing the identity:

$$LRCOND_k(\mathfrak{A}, [A \to \alpha.\pi], \mathfrak{B}, [B \to \beta.\delta], T) \equiv$$

$$LRCOND_k(\mathfrak{B}, [B \to \beta.\delta], \mathfrak{A}, [A \to \alpha.\pi], T)$$

$LRCOND_k$ has been defined for a pair of pairs (set of strings, item) and a state, (i.e. the domain: $\mathcal{P}(\Sigma^{*k}) \times$ item $\times\ \mathcal{P}(\Sigma^{*k}) \times$ item $\times$ state). Obviously $LRCOND_k$ may be generalized to have as arguments a pair of lists with elements (set of strings, item), and a state, corresponding to the domain: list of $(\mathcal{P}(\Sigma^{*k}) \times$ item$) \times$ list of $(\mathcal{P}(\Sigma^{*k}) \times$ item$) \times$ state.

4. AN ALGORITHM FOR LR(k) TESTING

In this section we will state an algorithm for LR(k)-testing based on the LR(0)-machine by using the results of the previous section. The theorems have been given a form such that a general scheme for an algorithm may be applied almost directly.

We may consider the identities in theorems 3.8 and 3.11 as a set of equations defining a recursive function $LRCOND_k$ from $\mathcal{P}(\Sigma^{*k}) \times$ item $\times\ \mathcal{P}(\Sigma^{*k}) \times$ item $\times$ state to $\{TRUE, FALSE\}$. We may solve this set of equations in order to decide if the grammar happens to be LR(k). The equations may have more than one solution but we are only interested in the smallest solution in the sense expressed by [4.1] and theorem 4.2 below.

Let $D_{COND-k} = \mathcal{P}(\Sigma^{*k}) \times$ item $\times\ \mathcal{P}(\Sigma^{*k}) \times$ item $\times$ state. Consider the function

$$F : D_{COND-k} \rightarrow \{TRUE, FALSE\}$$

given as a solution to the set of recursive equations:

[4.1]
$$F(\mathfrak{A}, [A \rightarrow \alpha\beta.\pi], \mathfrak{B}, [B \rightarrow \beta.\delta], T) \equiv$$
$$(BOTTOM_k(\mathfrak{B}, [B \rightarrow \beta.\delta], T) \Rightarrow$$
$$\mathfrak{A} \oplus_k LALR_k([A \rightarrow \alpha\beta.\pi], T) \cap \mathfrak{B} \oplus_k LALR_k([B \rightarrow \beta.\delta], T) = \emptyset)$$
$$\wedge$$
$$(\neg BOTTOM_k(\mathfrak{B}, [B \rightarrow \beta.\delta], T) \Rightarrow$$
$$\forall\ S \in PRED(T, \beta):$$
$$\forall\ [C \rightarrow \varphi.B\Psi] \in S:$$
$$F(\mathfrak{A}, [A \rightarrow \alpha.\beta\pi], \mathfrak{B} \oplus_k FIRST_k(\Psi), [C \rightarrow \varphi.B\Psi], S))$$

Let $TRUE < FALSE$ be the ordering on $\{TRUE, FALSE\}$.

Theorem 4.2

$LRCOND_k$ is the smallest solution to [4.1] in the sense that

if G is another solution to [4.1] then

$\forall (\mathbb{G}, I, \mathbb{B}, J, T) \in D_{COND-k}$:

$LRCOND_k(\mathbb{G}, I, \mathbb{B}, J, T) \leq G(\mathbb{G}, I, \mathbb{B}, J, T)$.

Proof: By the theorems 3.8 and 3.11 $LRCOND_k$ is a solution to [4.1].
We shall prove that, if G is any solution to [4.1] then:

$\forall (\mathbb{G}, I, \mathbb{B}, J, T) \in D_{COND-k}$:

$\neg\, LRCOND_k(\mathbb{G}, I, \mathbb{B}, J, T) \Rightarrow \neg\, G(\mathbb{G}, I, \mathbb{B}, J, T)$

Assuming that

$LRCOND_k(\mathbb{G}, I, \mathbb{B}, J, T) = FALSE$

the interesting case is

$BOTTOM_k(\mathbb{B}, J, T) = FALSE$

and we shall prove that

$G(\mathbb{G}, I, \mathbb{B}, J, T) = FALSE$.

In this case we have that

$\exists\ S \in URCORE_k(T)$:

$\exists\ x \in \Sigma^k$:

$x \in \mathbb{G} \oplus_k LR_k(I,S) \cap \mathbb{B} \oplus_k LR_k(J,S)$

such that $x = FIRST_k(yz) \wedge y \in \mathbb{G} \wedge z \in LR_k(I,S)$

and $x = FIRST_k(y'z') \wedge y' \in \mathbb{B} \wedge z' \in LR_k(J,S)$.

Assume that $I = [A \rightarrow \alpha\beta.\pi]$ and $J = [B \rightarrow \beta.\delta]$.

Consider any viable prefix τ, such that $GOTO_k(IS_k,\tau) = S$, i.e. $I, J \in V_k^G(\tau)$. We claim the existence of τ_1, φ_2, $\Psi_2, \omega \in (N \cup \Sigma)^*$ and $A_1 \rightarrow \varphi_1 A_2 \Psi_1 \in P$ such that:

$$S' \Rightarrow^* \tau_1 A_1 \omega \Rightarrow^* \tau_1 \varphi_1 A_2 \Psi_1 \omega \Rightarrow^* \tau_1 \varphi_1 \varphi_2 A \Psi_2 \Psi_1 \omega$$

where $\tau = \tau_1 \varphi_1 \varphi_2 \alpha\beta \wedge z \in FIRST_k(\Psi_2\Psi_1) \wedge z \notin FIRST_k(\Psi_2)$.

Similarly

$$S' \Rightarrow^* \tau_1' B_1 \omega' \Rightarrow^* \tau_1' \varphi_1' B_2 \Psi_1' \omega' \Rightarrow^* \tau_1' \varphi_1' \varphi_2' B \Psi_2' \Psi_1' \omega'$$

where $\tau = \tau_1' \varphi_1' \varphi_2' \beta \wedge z' \in FIRST_k(\Psi_2' \Psi_1') \wedge z' \notin FIRST_k(\Psi_2')$.

Let $\tau = X_1 X_2 \ldots X_n$ and $R_i = GOTO_k(IS_k, X_1 \ldots X_i)$ for $i = 1, \ldots, n$, $(R_n = S)$. Assume that $P(R_i) \in M_0$ is the projection of $R_i \in M_k$ on M_0 (i.e. $P(R_i) = CORE(R_i)$) for $i = 1, \ldots, n$. Finally assume that $R = GOTO_k(IS_k, \tau_1 \varphi_1)$ and $R' = GOTO_k(IS_k, \tau_1' \varphi_1')$. Using [4.1] for G, the $BOTTOM_k$-predicate will not have the value TRUE until either P(R) or P(R') is met by tracing backwards along $P(R_n), P(R_{n-1}), \ldots, P(R_1)$. $BOTTOM_k$ will at least be TRUE at IS_k. Assume that $BOTTOM_k$ becomes TRUE for parameters $(\mathfrak{A}', I', \mathfrak{B}', J', S')$ then

$$x \in \mathfrak{A}' \oplus_k LALR_k(I', S') \cap \mathfrak{B}' \oplus_k LALR_k(J', S')$$

which implies that $G(\mathfrak{A}, I, \mathfrak{B}, J, T) = FALSE$.

□

The algorithm for LR(k)-testing for $k \geq 1$ is realized by a function LRCOND-k as shown in algorithm 4.3. LRCOND-k has a local recursive procedure LRCOND which is a straightforward transcription of [4.1]. At each level of recursion a boolean value is AND'ed to a global boolean variable Q. A global variable DONE collects the actual parameter values of LRCOND in order to stop the recursion when a considered parameter value has occurred previously (i.e. is circularly dependent on itself).

Using theorem 3.2 the grammar G is then LR(k) if and only if

$\forall T \in M_0 : \forall [A \rightarrow \alpha.\pi], [B \rightarrow \beta.] \in T:$

the result of

LRCOND-k($[A \rightarrow \alpha.\pi]$, $[B \rightarrow \beta.]$, T)

is the value TRUE.

A proof of the correctness of algorithm 4.3 may be based on a general algorithm for solving a set of recursive equations given in [Kristensen & Madsen 79a] and the theorems 3.8, 3.11 and 4.2. Unfortunately this general algorithm is expressed in terms of sets and not in terms of booleans. A straightforward solution to this problem is to implement boolean by a set:

Booleanset = SET OF {FALSE, TRUE}.

Let Q, QQ be of type boolean and let Q', QQ' be the corresponding elements of type Booleanset, then

TRUE/FALSE is implemented as {TRUE}/{FALSE},
$Q \wedge QQ$ is implemented as $Q' \cup Q' Q'$,
Q' may be interpreted as a boolean with the value FALSE $\notin$ Q'.

Notation
In the algorithm the construct

ASSUME ... ;
is used for namegiving of (components of) structured variables.

FOR a $\in$ M WHERE P_a DO S ENDFOR;
means
FOR a $\in$ M DO
IF P_a THEN S ENDIF
ENDFOR;

□

Algorithm 4.3

D_{COND-k} = SET OF Σ^{*k} x item x SET OF Σ^{*k} x item x state;

FUNCTION LRCOND-k (I, J: item; T: state): boolean;

VAR

Q: boolean;

DONE : SET OF D_{COND-k};

PROCEDURE LRCOND($\mathbb{A}$, I, $\mathbb{B}$, J, T): D_{COND-k});

BEGIN

DONE := DONE $\cup$ {($\mathbb{A}$, I, $\mathbb{B}$, J, T)};

IF $BOTTOM_k(\mathbb{A}, I, T) \vee BOTTOM_k(\mathbb{B}, J, T)$ THEN

$Q := Q \wedge (\mathbb{A} \oplus_k LALR_k(I, T) \cap \mathbb{B} \oplus_k LALR_k(J, T) = \emptyset)$

ELSE

ASSUME $I = [A \to \alpha\beta.\pi]$, $J = [B \to \beta.\delta]$;

FOR $S \in PRED(T, \beta)$ DO

FOR $[C \to \varphi.B\Psi] \in S$

WHERE ($\mathbb{A}$, $[A \to \alpha.\beta\pi]$, $\mathbb{B} \oplus_k FIRST_k(\Psi)$,

$[C \to \varphi.B\Psi]$, S) $\notin$ DONE DO

LRCOND($\mathbb{A}$, $[A \to \alpha.\beta\pi]$, $\mathbb{B} \oplus_k FIRST_k(\Psi)$, $[C \to \varphi.B\Psi]$, S)

ENDFOR;

ENDFOR;

ENDIF;

END LRCOND;

BEGIN

Q := true; DONE := $\emptyset$;

ASSUME $I = [A \to \alpha.\pi]$, $J = [B \to \beta.]$;

LRCOND($EFF_k(\pi)$, I, {e}, J, T);

LRCOND-k := Q;

END LRCOND-k; □

Algorithm 4.3 uses the function $LALR_k$ to compute LALR(k)-lookahead. The LALR(k)-lookahead may be computed when needed - or in advance. Algorithms for this are given in [Kristensen & Madsen 79a]. In section 5 we discuss a possibility for integrating the LALR(k) computation in the LRCOND-k computation.

We remark that the k used in the $LALR_k$ computation may be decreased by the length of the shortest string in the first operand of the $\oplus_k$-operator (i.e. $\mathbb{A}$ or $\mathbb{B}$). Furthermore $BOTTOM_k(\mathbb{A}, I, T)$ implies that $\mathbb{A} \oplus_k LALR_k(I, T)$ may be computed by only considering state T (i.e. if $|\mathbb{A}|_{min} = i$ then T contains the necessary information to compute $LALR_{k-i}(I, T)$).

A trivial upper bound on the number of recursive calls of procedure LRCOND in algorithm 4.3 is the size of the domain for DONE, D_{COND-k}, that is

$$\mathcal{O}(|\mathcal{P}(\Sigma^{*k})|^2 \times |\text{item}|^2 \times |\text{state}|)$$

This may be improved to

$$[4.4] \qquad \mathcal{O}(|\mathcal{P}(\Sigma^{*k})|^2 \times \sum_{T \in M_0} (\#\ \text{items in } T)^2)$$

as LRCOND is only activated for pairs of items in a state T.

For practical grammars the size of Σ means that [4.4] is unusable; however for small test grammars the bound may be reached.

In section 5 we discuss another approach with an improved performance. In any case the difference between the upper bounds and the normal case in practical situations is still very large.

5. PRACTICAL REMARKS AND IMPROVEMENTS

This section describes how to utilize the algorithm for LR(k)-testing in an LALR(k) parser generator system as a means of giving informative diagnostics on LALR(k)-conflicts. Furthermore we describe alternative versions and suggest various improvements of the algorithm for testing LR-conditions.

5.1 Informative Diagnostics on LALR(k)-Conflicts

Let $I = [A \to \alpha.\pi]$, $J = [B \to \beta.]$ be items in a state $T \in M_0$. If

$$[5.1] \qquad EFF_k(\pi) \oplus_k LALR_k(I,T) \cap LALR_k(J,T) \neq \emptyset$$

a LALR(k)-conflict exists for I, J in T. In this case we may as a minimum inform about the conflict by giving the sets appearing in [5.1], the items I, J and state T.

A more informative diagnostic will be to give the set of viable prefixes leading to T. If we consider the recursive formulation of LALR(k) in theorem 2.9 then each viable prefix may be characterized by a list of pairs (item, state) obtainable by tracing backwards in the recursive definition of $LALR_k(I,T)$. Each viable prefix may be characterized by several lists. Only a prefix-part of a given list gives a lookahead contribution to $LALR_k(I,T)$. Let TRACE(I, T) be the set of all such prefixes.

Each trace, $L \in TRACE(I,T)$, defines a lookahead contribution (called LA(L)) to the set $LALR_k(I,T)$ and a path (called PATH(L)) starting in T and leading backwards in the predecessor tree of T. We have that

$$[5.2] \qquad LALR_k(I,T) = \cup\{LA(L) \mid L \in TRACE(I,T)\}$$

The set of traces (and viable prefixes) may be infinite. The LALR(k) algorithm in [Kristensen & Madsen 79a] traverses a finite number of traces. These traces may easily be collected (in e.g. a stack) and given as informative diagnostics.

Furthermore, if

[5.3] $\exists R \in URCORE_k(T)$:

$$EFF_k(\pi) \oplus_k LR_k(I,R) \cap LR_k(J,R) \neq \emptyset$$

then the LALR(k)-conflict is also an LR(k)-conflict. If there is no LR(k)-conflict, i.e. [5.3] does not hold, it may be problematic to understand the LALR(k)-conflict and to repair it, using only the above described information. This is because the conflict appears as a result of the implicit merging process of the states $R \in M_k$ such that $R \in URCORE_k(T)$, cf. definition 2.5. A better diagnostic is to give the LR(k)-sets for I, J in $URCORE_k(T)$, i.e. the sets appearing in [5.3].

As above the diagnostics may be improved by giving for each R in $URCORE_k(T)$ the set of viable prefixes leading to R. (Note that the viable prefixes leading to R in M_k are a subset of the viable prefixes leading to T in M_0).

This can be done by grouping the viable prefixes leading to T and consequently grouping the traces of (I, T).

A set of traces, $M \subseteq TRACE(I,T)$ characterizes viable prefixes leading to the same state in $URCORE_k(T)$ if for all $L_1, L_2 \in M$, $PATH(L_1)$ is a prefix of $PATH(L_2)$ or vice versa (denoted $PATH(L_1) \sim PATH(L_2)$). This gives us that if M is maximal, i.e.

[5.4] $$\forall L_1 \in TRACE(I,T) : (L_1 \in M \Leftrightarrow (\forall L_2 \in M : PATH(L_1) \sim PATH(L_2)))$$

then $\exists R \in URCORE_k(T)$:

$$LR_k(I,T) = \cup\{LA(L) \mid L \in M\}$$

Consider now

[5.5] $L_I \in TRACE(I, T)$ and

$L_J \in TRACE(J, T)$

If $PATH(L_I) \sim PATH(L_J)$ then L_I and L_J represent viable prefixes leading to the same state, $R \in URCORE_k(T)$ and $LA(L_I) \subseteq LR_k(I, R)$ and $LA(L_J) \subseteq LR_k(J, R)$.

Information about traces for (I, T) and (J, T) described by [5.5] will thus be sufficient to inform about viable prefixes leading to states in $URCORE_k(T)$.

These traces may be obtained using $LRCOND_k$. The recursive definition of $LRCOND_k$ defines a set of traces in the same way as is done for LALR(k). Each trace defined by $LRCOND_k$ defines a unique trace $L_I \in TRACE(I, T)$ and a unique trace $L_J \in TRACE(J, T)$ such that $PATH(L_I) \sim PATH(L_J)$.

Again only a finite set of traces need to be considered in order to have sufficient information about all traces and the LRCOND-k algorithm (4.3) generates such a subset.

LRCOND-k stops when it has a trace for either (I, T) or (J, T). This happens when the predicate

[5.6] $BOTTOM_k(\mathbb{G}, I', T') \vee BOTTOM_k(\mathbb{B}, J', T')$

becomes true.

If $BOTTOM_k(\mathbb{G}, I', T')$ becomes true then we have a trace for e.g. (I, T). We do not necessarily have a trace for (J, T) but we have a prefix part of one or more traces in TRACE(J,T). If $I = [A \rightarrow \alpha . \pi]$ then the initial call is LRCOND-k $(EFF_k(\pi), I, \{e\}, J, T)$. Because of $EFF_k(\pi)$ we may have $BOTTOM_k$ before a complete trace for (I, T) is generated. In all cases we have sufficient information for testing LR(k).

For informative reasons it may be desirable to generate the remaining parts of the traces. This can be done by the succeeding call of $LALR_k(J', T)$ or by continuing the recursion until we have $BOTTOM_k$ for both arguments (change $\vee$ to $\wedge$ in [5.6]). In this connection one should also consider the integration of LALR(k) computation in LRCOND-k, cf. section 5.2.

Besides informing about a given trace one may give more detailed information about the trace, e.g. for each item in a trace, L, give the subset of LA(L) coming from that item.

To summarize: LRCOND-k may be used to generate corresponding pairs of traces $L_I \in TRACE(I, T)$, $L_J \in TRACE(J, T)$ representing viable prefixes leading to the same state in $URCORE_k(T)$. Furthermore it is easy to indicate where possible LR(k) or LALR(k) conflicts appear. This information seems to be sufficient informative diagnostics and the only remaining problem may be to reduce the amount of information.

5.2 Integration of LALR(k)-Lookahead Computation in Algorithm 4.3

We have discussed how to apply the LRCOND-k algorithm in connection with a system computing LALR(k)-lookahead.

Another possibility is to integrate the LALR(k)-lookahead computation directly in algorithm 4.3 based on the observation that the same predecessor tree is traversed in the two computations. The formula for computing LALR(k)-lookahead is given by theorem 2.9. An algorithm testing LR-conditions which automatically collects LALR-lookahead is included in Appendix B.

5.3 The k = 1 Case

In practice the case k = 1 is the most important. We shall improve theorem 3.11 for k = 1.

Theorem 5.7

Let $T \in M_0$ and $[A \to \alpha\beta.\pi]$, $[B \to \beta.\delta] \in T$ then

$$LRCOND_1(\{e\}, [A \to \alpha\beta.\pi], \{e\}, [B \to \beta.\delta], T) \equiv$$
$$\forall S \in PRED(T,\beta):$$
$$(\forall [C \to \varphi.B\Psi] \in S \text{ where } \Psi \Rightarrow^* e:$$
$$LRCOND_1(\{e\}, [A \to \alpha.\beta\pi], \{e\}, [C \to \varphi.B\Psi], S))$$
$$\wedge$$
$$(LALR_1([A \to \alpha.\beta\pi], S) \cap (\cup\{FIRST_1(\Psi) \mid [C \to \varphi.B\Psi] \in S\} \setminus \{e\}) = \emptyset)$$

□

An algorithm similar to algorithm 4.3, constructed directly from theorem 5.7 is shown in Appendix B.

The set $\cup\{FIRST_1(\Psi) \mid [C \to \varphi.B\Psi] \in S\} \setminus \{e\}$ may be computed directly on LRM_0 starting in the state $GOTO_0(S,B)$, by an algorithm given in [Kristensen & Madsen 79a].

An algorithm based on theorem 5.7 has been integrated in an LALR(1) parser generator system, the BOBS-system, [Eriksen et al. 73] with reference to better diagnostics on LALR(1) conflicts. Consider two conflicting items I, J in a state T. It is possible to obtain the following information:

(1) The conflicting symbols in $LALR_1(I,T)$ and $LALR_1(J,T)$.

(2) A marking of those symbols (if any) which cause an LALR conflict but which do not cause an LR-conflict between I and J in any state in $URCORE_1(T)$.

(3) Parts of the traces obtainable by tracing backwards with I, J from T using $LRCOND_1$. At each item the conflicting symbols generated by the item are given. Only traces that generate conflicting symbols are given.

(4) All items in T including their LALR(1)-lookahead.

(5) The complete set of traces generated by $LRCOND_1$ as described in (3).

(6) The whole LR(0)-machine.

In many cases (1) and (2) are sufficient, but often (3) is needed. (4), (5) and (6) are seldom necessary and especially (6) may produce an enormous amount of output. So far these diagnostics are very satisfactory.

5.4 Improvement of Algorithm 4.3

In this section an alternative approach with a better worst case performance is given. Instead of the predicate $LRCOND_k$, we base the computation on $FSETS_{m,n}$ which defines a set of pairs of sets, to be used in LR(k)-testing. Let

[5.8]
$$FSETS_{m,n}([A \to \alpha\beta.\pi], [B \to \beta.\delta], T) =$$
$$\cup\{\{(\{e\}, FIRST_n(\Psi))\} \boxplus_{m,n} FSETS_{m,n}([A \to \alpha.\beta\pi], [C \to \varphi.B\Psi], S)$$
$$\mid S \in PRED(T,\beta) \wedge [C \to \varphi.B\Psi] \in S\}$$

such that

$$FSETS_{m,n} : item \times item \times state \to sets\ of\ (\Sigma^{*m} \times \Sigma^{*n}).$$

(Note that $FSETS_{m,n}(I,J,T) = \{(\mathbb{G},\mathbb{B}) \mid (\mathbb{B},\mathbb{G}) \in FSETS_{n,m}(J,I,T)\}$).

Notation

Let $\mathfrak{T},\mathfrak{S}$ be sets of $(\Sigma^{*m} \times \Sigma^{*n})$ then

$$\mathfrak{T} \boxplus_{m,n} \mathfrak{S} =$$
$$\{(T_1 \oplus_m S_1,\ T_2 \oplus_n S_2) \mid (T_1, T_2) \in \mathfrak{T} \wedge (S_1, S_2) \in \mathfrak{S}\}$$

□

Let $LRSETS_{m,n}$ denote the minimal solution to [5.8].

It may be shown that for $T \in M_0$ and $I, J \in T$:

[5.9] $$LRCOND_k(\{e\}, I, \{e\}, J, T) \equiv$$
$$\forall (\mathfrak{A}, \mathfrak{B}) \in LRSETS_{k,k}(I, J, T): \mathfrak{A} \cap \mathfrak{B} = \emptyset$$

Thus we may base an alternative algorithm on $LRSETS_{m,n}$.

$LRSETS_{k,k}$ may be related to $LRCOND_k$ in the following way: $LRCOND_k$ generates corresponding pairs of traces (L_I, L_J) described by [5.5]. $LRSETS_{k,k}$ generates pairs of sets $(LA(L_I), LA(L_J))$ where L_I, L_J are described by [5.5], i.e.

$$LRSETS_{k,k}(I, J, T) =$$
$$\{(LA(L_I), LA(L_J)) \mid L_I \in TRACE(I, T) \wedge L_J \in TRACE(J, T)$$
$$\wedge PATH(L_I) \sim PATH(L_J)\}$$

Unfortunately the general algorithm from [Kristensen & Madsen 79a] cannot be applied directly for [5.8]. It is straightforward to reformulate [5.8] to be able to apply the scheme for the construction of the corresponding algorithm. This transformation follows the same ideas as for $LALR_k$ as presented in [Kristensen & Madsen 79a]. Using this approach we find an upper bound of the number of recursive calls in the computation of the form

$$\mathcal{O}(K_2(M_0) + 2(k-1)K_2(M_0)^2)$$

where

$$K_2(M_0) = \sum_{T \in M_0} (\#\text{ items in } T)^2$$

The reformulation of $LRSETS_{m,n}$ and the corresponding algorithm are given in Appendix B.

5.5 Generalizing Algorithm 4.3 to compute $LRCOND_k$ for Parametersets of Internal Recursive Calls

Algorithm 4.3 computes $LRCOND_k$ for the actual parameterset of the initial call of LRCOND. We may generalize the general scheme to compute $LRCOND_k$ for any actual parameterset appearing in the internal recursive calls of LRCOND. The general algorithm supporting this facility is discussed in [Kristensen & Madsen 79a]. The number of recursive calls of LRCOND will not be changed if algorithm 4.3 is transformed into this general scheme.

The approaches outlined previously in this section may also be generalized to this scheme.

An improved version of the general algorithm is proved in [Kristensen & Madsen 79b], which also includes the algorithm for testing the LR-condition for $k = 1$, generalized to this scheme.

6. REFERENCES

Aho, A.V. & Ullman, J.D. [1972, 1973]
"The Theory of Parsing, Translation and Compiling"
Vol. I & II, Prentice-Hall, Englewood Cliffs, N.J., 1973.

DeRemer, F.L. [1969]
"Practical Translators for LR(k) Languages"
Ph.D. Diss., MIT, Cambridge, Mass., 1969.

DeRemer, F.L. [1971]
"Simple LR(k) Grammars"
Comm. ACM 14:7, 453-460, 1971.

Eriksen, S.H., Jensen, B.B., Kristensen, B.B. & Madsen, O.L. [1973]
"The BOBS-system"
Computer Science Department, Aarhus University, 1973.
(Revised version DAIMI PB-71, 1979).

Hunt, H.B., Szymanski, T.G. & Ullman, J.D. [1975]
"On the Complexity of LR(k) Testing"
Comm. ACM 18:12, 707-716, 1975.

Knuth, D.E. [1965]
"On the Translation of Languages from Left to Right"
Info. Contr., 8, 6 (1965), 607-639.

Kristensen, B.B. & Madsen, O.L. [1979a]
"Methods for Computing LALR(k) Lookahead"
Computer Science Department, Aarhus University, 1979.
DAIMI PB-101.

Kristensen, B.B. & Madsen, O.L. [1979b]
"Correctness Proof of a General Algorithm for Solving a Set of Recursive Equations (Exemplified by Algorithms Based on LALR(k) and LR(k) Theory)"
Computer Science Department, Aarhus University, 1979.

Pager, D. [1977a]
"The Lane-Tracing Algorithm for Constructing LR(k) Parsers and Ways of Enchancing its Efficiency"
Inf. Sci. 12, 19-42 (1977).

Pager, D. [1977b]
"A Practical General Method for Constructing LR(k) Parsers"
ACTA Informatica 7, 249-268 (1977).

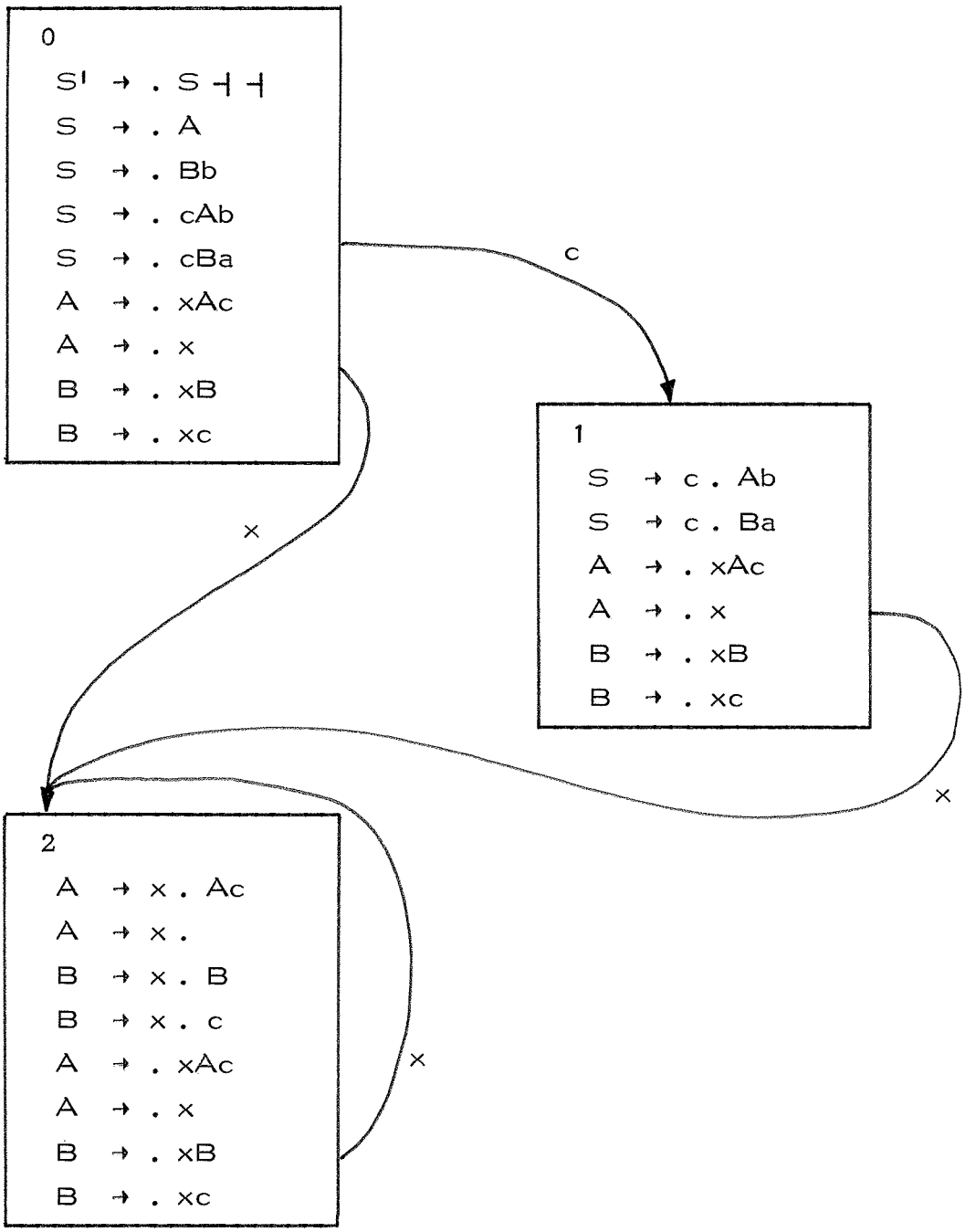
Appendix A: Example

Let $G_1 = (\{S', S, A, B\}, \{a, b, c, x, \dashv\}, P, S')$ be a CFG, where P consists of:

$S' \rightarrow S \dashv \dashv$
$S \rightarrow A \mid Bb \mid cAb \mid cBa$
$A \rightarrow xAc \mid x$
$B \rightarrow xB \mid xc$

Using G_1 we may show the correspondence between the LR(2)-machine for G_1, LRM_2^{G1}, - and the LR(2)-testing by $LRCOND_2$ using the LR(0)-machine for G_1, LRM_0^{G1}.

The involved part of LRM_0^{G1} can be illustrated as


0
S' → . S ⊣ ⊣
S → . A
S → . Bb
S → . cAb
S → . cBa
A → . xAc
A → . x
B → . xB
B → . xc
c
x
1
S → c . Ab
S → c . Ba
A → . xAc
A → . x
B → . xB
B → . xc
x
2
A → x . Ac
A → x .
B → x . B
B → x . c
A → . xAc
A → . x
B → . xB
B → . xc
x

G_1 is $\neg$ LALR(2) as

$$LALR_2([A \to x.\,], 2) = \{\dashv\dashv, b\dashv, c\dashv, cb, cc\}$$

and

$$FIRST_2(c) \oplus_2 LALR_2([B \to x.\,c], 2) = \{cb, ca\}.$$

Consider the following computation:

$$LRCOND_2(\{c\}, [B \to x.\,c], \{e\}, [A \to x.\,], 2) \equiv$$

[A.1] $LRCOND_2(\{c\}, [B \to .\,xc], \{e\}, [S \to .\,A], 0) \wedge$

[A.2] $LRCOND_2(\{c\}, [B \to .\,xc], \{b\}, [S \to c.\,Ab], 1) \wedge$

[A.3] $LRCOND_2(\{c\}, [B \to .\,xc], \{c\}, [A \to x.\,Ac], 2)$

For the cases [A.1], [A.2], [A.3] we find that:

$$[A.1] \equiv LRCOND_2(\{cb\}, [S \to .\,Bb], \{e\}, [S \to .\,A], 0)$$

where $BOTTOM_2(\{e\}, [S \to .\,A], 0) \equiv$ TRUE and
$\{e\} \oplus_2 LALR_2([S \to .\,A], 0) = \{\dashv\dashv\}$ implies

[A.4] $[A.1] \equiv \{cb\} \cap \{\dashv\dashv\} = \emptyset \equiv$ TRUE.

$$[A.2] \equiv LRCOND_2(\{ca\}, [S \to c.\,Ba], \{b\}, [S \to c.\,Ab], 1)$$

where $BOTTOM_2(\{ca\}, [S \to c.\,Ba], 1) \equiv$ TRUE and
$\{b\} \oplus_2 LALR_2([S \to c.\,Ab], 1) = \{b\dashv\}$ implies

[A.5] $[A.2] \equiv \{ca\} \cap \{b\dashv\} = \emptyset \equiv$ TRUE.

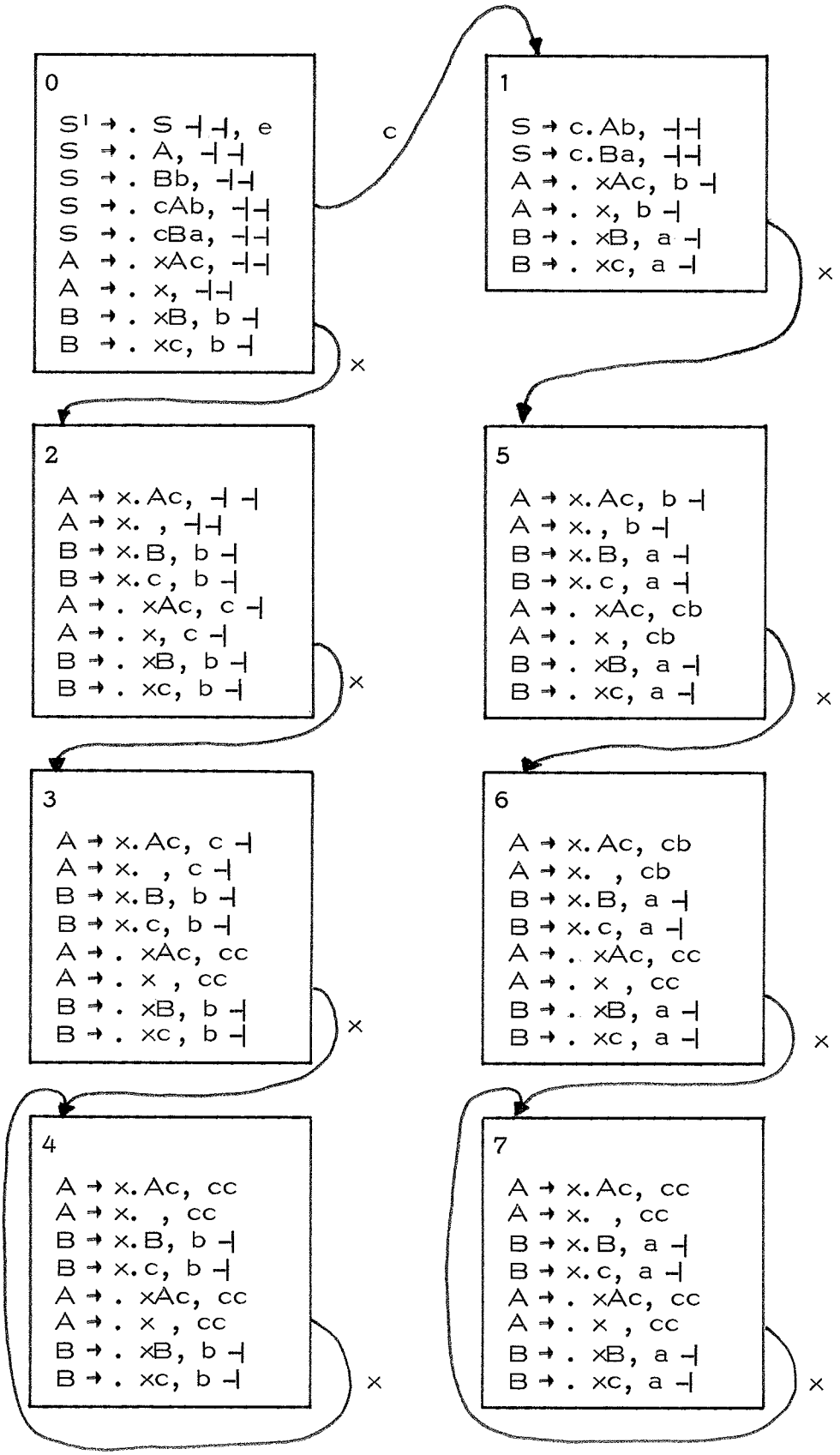
Similarly we find that [A.3] $\equiv$

[A.6] $\{cb\} \cap \{c\dashv\} = \emptyset \wedge$

[A.7] $\{ca\} \cap \{cb\} = \emptyset \wedge$

[A.8] $\{ca, cb\} \cap \{cc\} = \emptyset \equiv$ TRUE.

The involved part of LRM_2^{G1} can be illustrated as:



To test if G_1 is LR(2), concerning the items $[A \to x.]$ and $[B \to x.c]$, we must compare

(a) $LR_2([A \to x.], T)$ and

(b) $\{c\} \oplus_2 LR_2([B \to x.c], T)$

for T = 2, 3, 4, 5, 6, 7 :

T	(a)	(b)	(c)
2	⊣ ⊣	cb	[A.4]
3	c ⊣	cb	[A.6]
4	cc	cb	[A.8]
5	b ⊣	ca	[A.5]
6	cb	ca	[A.7]
7	cc	ca	[A.8]

The column (c) shows the correspondence between the test based on the LRM_2^{G1} and the computation using $LRCOND_2$.

□

Appendix B: Algorithms for Alternative Approaches

B. 1 Algorithm on LR-conditions for the k = 1 case.
B. 2 Improved algorithm for testing the LR-condition.
B. 3 Algorithm with integrated LALR-lookahead computation.

B. 1 Algorithm on LR-conditions for the k=1 case

The algorithm for the k=1 case is realized by a function LRCOND-1 as shown in algorithm B. 1. The algorithm is a straightforward transcription of theorem 5. 7 and the following well-known result:

$$
\begin{array}{l}
(\exists\, T \in M_0: \\
\quad \exists\, [A \to \alpha.\pi],\ [B \to \beta.] \in T: \\
\qquad (EFF_1(\pi)\setminus\{e\}) \cap LALR_1([B \to \beta.],\ T) \neq \emptyset\ ) \\
\quad \Downarrow \\
\qquad G \text{ is not LR(1).}
\end{array}
$$

A proof of the correctness of the algorithm may be based on theorem 4. 2 and the general scheme in [Kristensen & Madsen 79a]. An upper bound on the number of calls of the procedure LRCOND is

$$\mathcal{O}\Big(\sum_{T \in M_0} (\#\text{ items in } T)^2\Big)$$

Algorithm B.1

D_{COND-1} = item × item × state;
FUNCTION LRCOND-1 ((I, J, T) : D_{COND-1}): boolean;

VAR
Q : boolean;
DONE: SET OF D_{COND-1};
PROCEDURE LRCOND((I, J, T) : D_{COND-1});
BEGIN
DONE := DONE ∪{(I, J, T)};
ASSUME I = [A → αβ.π], J = [B → β.δ];
FOR S ∈ PRED(T, β) DO
Q := Q ∧ ($LALR_1$([A → α.βπ], S) ∩
(∪{$FIRST_1$(Ψ) | [C → φ.BΨ] ∈ S}\{e}) = ∅);
FOR [C → φ.BΨ] ∈ S DO
ASSUME I' = [A → α.βπ], J' = [C → φ.BΨ];
IF e ∈ $FIRST_1$(Ψ) ∧ (I', J', S) ∉ DONE THEN
LRCOND(I', J', S);
ENDIF;
ENDFOR;
ENDFOR;
END LRCOND;

BEGIN
ASSUME I = [A → α.π], J = [B → β.];
Q := (EFF_1(π)\{e}) ∩ $LALR_1$(J, T) = ∅;
DONE := ∅;
LRCOND(I, J, T);
LRCOND-1 := Q;
END LRCOND-1;

□

B.2 Improved Algorithm for Testing the LR-Condition

In order to be able to apply the general algorithm in [Kristensen & Madsen 79a] we need to reformulate $FSETS_{m,n}$ as given by the following definition.

Definition B.2

Let $T \in M_0$ and $[A \rightarrow \alpha\beta.\pi]$, $[B \rightarrow \beta.\delta] \in T$ then

[B.3]
$$FTESTSETS_{m,n}([A \rightarrow \alpha\beta.\pi], [B \rightarrow \beta.\delta], T) =$$
$$\cup\{F_{m,n}(I,B,S) \cup FL_{m,n}(I,B,S) \cup L_{m,n}(I,B,S) \mid S \in PRED(T,\beta)\}$$

where $I = [A \rightarrow \alpha.\beta\pi]$ and

$$F_{m,n}(I,B,S) = \{(LALR_m(I,S), \{w \mid w \in FIRST_n(\Psi) \wedge |w| = n\}) \mid [C \rightarrow \varphi.B\Psi] \in S\},$$

$$FL_{m,n}(I,B,S) = \{\{(\{e\}, FIRST_n(\Psi)\setminus\{e\})\} \boxplus_{m,n} FTESTSETS_{m,i}(I,[C \rightarrow \varphi.B\Psi], S) \mid [C \rightarrow \varphi.B\Psi] \in S \wedge i = n - |FIRST_n(\Psi)\setminus\{e\}|_{min} \wedge 0 < i \leq n\},$$

$$L_{m,n}(I,B,S) = \{FTESTSETS_{m,n}(I,[C \rightarrow \varphi.B\Psi], S) \mid [C \rightarrow \varphi.B\Psi] \in S \wedge \Psi \Rightarrow^* e\}$$

Note that $FTESTSETS_{m,n}(I,J,T) = \{(\mathbb{G},\mathbb{B}) \mid (\mathbb{B},\mathbb{G}) \in FTESTSETS_{n,m}(J,I,T)\}$. Let $LRTESTSETS_{m,n}$ be the minimal solution to the recursive equations defined by [B.3].

□

The correspondence between $FSETS_{m,n}$ and $FTESTSETS_{m,n}$ may be described by introducing a minor change in [B.3], namely:

[B.4]
$$F'_{m,n}(I,B,S) = \{(\{e\}, \{w \mid w \in FIRST_n(\Psi) \wedge |w| = n\}) \boxplus_{m,n} FTESTSETS_{m,0}(I,[C \rightarrow \varphi.B\Psi], S) \mid [C \rightarrow \varphi.B\Psi] \in S\}.$$

Using [B.4] we may obtain

- $FTESTSETS_{m,n}$ by constructing $F_{m,n}$ as the union of elements coming from $F'_{m,n}$,
- $FSETS_{m,n}$ as the union of the elements coming from $F'_{m,n}$, $FL_{m,n}$ and $L_{m,n}$ for the same item $[C \to \varphi.B\Psi]$ in S.

Theorem B.5

Let $T \in M_0$ and $I, J \in T$ then

$$LRCOND_k(\{e\}, I, \{e\}, J, T) \equiv$$
$$\forall (\mathfrak{A}, \mathfrak{B}) \in LRTESTSETS_{k,k}(I, J, T) : \mathfrak{A} \cap \mathfrak{B} = \emptyset$$

□

Using definition B.2 we may now construct the algorithm for testing the LR-condition. It is realized by the function LRTESTSETS-m-n as shown in algorithm B.6. LRTESTSETS-m-n has a local recursive procedure which computes $LRTESTSETS_{m,n}$ for fixed m and n. A proof of the correctness of algorithm B.6 may be based on theorem B.5 and the general algorithm of [Kristensen & Madsen 79a].

An upper bound for the number of recursive calls of procedure LRTESTSETS for each invocation of LRTESTSETS-m-n is

$$\mathcal{O}(K_2(M_0)).$$

Thus the total number for all invocations of LRTESTSETS-m-n(I, J, T, m, n) has the upper bound

$$\mathcal{O}(K_2(M_0)^{m+n-1}).$$

By saving the values computed by LRTESTSETS-m-n the upper bound reduces to

$$\mathcal{O}(K_2(M_0) + (m+n-2)\, K_2(M_0)^2)$$

for one call of LRTESTSETS-m-n(I, J, T, m, n) and to

$$\mathcal{O}((m+n-1)\, K_2(M_0)^2)$$

for the calls of LRTESTSETS-m-n(I, J, T, m, n) for all I, J, T.

Algorithm B.6

```
D_TEST = SET OF item x item x state;
D = SET OF (SET OF Σ*^m x SET OF Σ*^n);

FUNCTION LRTESTSETS-m-n((I, J, T) : D_TEST; m, n: Integer): D;

VAR SETS : D; Done : D_TEST;
    PROCEDURE LRTESTSETS((I, J, T) : D_TEST);
    VAR F : SET OF Σ*^k; i : Integer;
    BEGIN
        Done := Done ∪ {(I, J, T)};
        ASSUME I = [A → αβ.π], J = [B → β.δ];
        FOR S ∈ PRED(T, β) DO
            FOR [C → φ.BΨ] ∈ S DO
                ASSUME I' = [A → α.βπ], J' = [C → φ.BΨ];
                F := FIRST_n(Ψ);
                SETS := SETS ∪ {(LALR_m(I', S), {w ∈ F | |w| = n})};
                i := n - |F \ {e}|_min;
                IF 0 ≤ i < n THEN
                        SETS := SETS ∪ {({e}, F\{e})}⊞_m,n
                                LRTESTSETS-m-n(I', J', S, m, i);
                ENDIF;
                IF e ∈ F ∧ (I', J', S) ∉ Done THEN LRTESTSETS(I', J', S);
                ENDIF;
            ENDFOR;
        ENDFOR;
    END LRTESTSETS;

BEGIN
    SETS := Done := ∅;
    LRTESTSETS(I, J, T);
    LRTESTSETS-m-n := SETS;
END LRTESTSETS-m-n;
```

In the important special case where m = n = 1 [B.3] reduces straightforward and an improved algorithm is obtainable.

B.3 Algorithm with Integrated LALR-Lookahead Computation

The LALR-lookahead computation is integrated in the algorithm for LR(k) testing based on $LRCOND_k$ for k = 1. The approach is realized by a function LRLALR-1 as shown in algorithm B.7. LRLALR-1(I, J, T) computes ($LRCOND_1$(I, J, T), $LALR_1$(I, T), $LALR_1$(J, T)). A similar algorithm may be achieved by integration based on the algorithm for computing $LRTESTSETS_{m,n}$.

Notation

In the algorithm the construct

```
FOR a ∈ M WHERE P_a DO S_1
ELSEDO S_2
ENDFOR;
```

means

```
B := true;
FOR a ∈ M DO
    IF P_a THEN
        B := false;
        S_1
    ENDIF;
ENDFOR;
IF B THEN S_2 ENDIF;
```

□

Algorithm B.7

D_{COND} = item × item × state;

FUNCTION LRLALR-1((I, J, T): D_{COND}): boolean × SET OF Σ × SET OF Σ;
VAR Q : boolean; Done: SET OF D_{COND};
    LALRIT, LALRJT: SET OF Σ;
    PROCEDURE LRLALR((I, J, T): D_{COND}; VAR LIT, LJT: SET OF Σ);
    VAR LI'S, LJ'S, LJ''S, F : SET OF Σ;
    BEGIN
        Done := Done ∪ {(I, J, T)}
        ASSUME I = [A → αβ.π], J = [B → β.δ];
        ASSUME I' = [A → α.βπ], J' = [B → .βδ];
        LIT := LJT := ∅;
        FOR S ∈ PRED(T, β) DO
            LI'S := LJ'S := ∅;
            FOR [C → φ.BΨ] ∈ S WHERE Ψ ⇒* e DO
                ASSUME J'' = [C → φ.BΨ];
                IF (I', J'', S) ∉ DONE THEN LRLALR((I', J'', S), LI'S, LJ''S);
                        LJ'S := LJ'S ∪ LJ''S;
                ELSE LJ'S := LJ'S ∪ $LALR_1$(J'', S)
                ENDIF;
            ELSEDO LI'S := $LALR_1$(I', S);
            ENDFOR;
            F := ∪{$FIRST_1$(Ψ) | [C → φ.BΨ] ∈ S}\{e};
            LJ'S := LJ'S ∪ F;
            LJT := LJT ∪ LJ'S;
            LIT := LIT ∪ LI'S;
            Q := Q ∧ (LI'S ∩ F = ∅);
        ENDFOR;
    END LRLALR;

BEGIN
    Q := true; Done := ∅;
    LRLALR(I, J, T, LALRIT, LALRJT);
    ASSUME I = [A → α.π], J = [B → β.];
    Q := Q ∧ ((EFF_1(π)\{e}) ∩ LALRJT = ∅);
    LRLALR-1 := (Q, LALRIT, LALRJT);
END LRLALR-1;

□

We remark in algorithm B.7 that

- the parameter LI'S in LRLALR((I',J'',S),LI'S,LJ''S) delivers the same result, $LALR_1(I',S)$, for all possible J'' for fixed S. This is a result of the so far missing generalization of the concepts to be defined for a pair of lists of items and not just for a pair of items,
- in the case where

  $$\nexists [C \to \varphi . B \Psi] \in S : \Psi \Rightarrow^* e$$

  for a given B, and in the case where

  $$(I', J'', S) \in \text{Done}$$

  the recursion is not continued. Thus we miss the automatic collection of LALR(1)-lookahead and have to compute this separately by a function $LALR_1$.