A YACC EXTENSION FOR LRR GRAMMAR PARSING

B. SEITE
B.P. 145, 93163 Noisy Le Grand Cedex, France

Communicated by M. Nivat
Received May 1986

Abstract. The algorithm we present here generates finite-state automata for potentially unbounded examination of lookahead and stack in order to try to discriminate the conflicting actions of an LR0 collection. Furthermore, this algorithm can be considered as an overall model to extend the current LR parser generators preserving (i): the valid prefix property and the traditional error recovery routines of the LR parser; (ii): the LALR(1) context on the LR0 collection. The power of acceptance is a subset of Cohen-Culik's LRR, with acceptance of non-LR(k) grammars, allowing a deterministic bottom-up parsing in linear time when this succeeds.

The special feature of this method, compared to foregoing endeavours, consists in the use of the stack language so that a good deal of LR(k) grammars not accessible by foregoing methods now becomes acceptable. In terms of practicability, the minimization techniques allow one to get very compact automata as illustrated on the output lists. The generation of complementary tables can be done independently of the parser generator, which makes the connection of this complementary module to any LR parser generator quite easy. We also show some original results regarding LR automata.

Introduction

The class of acceptance of the current parser generators is quite limited. For the most powerful ones among them, when using the deterministic bottom-up LR parsing, the class of acceptance coincides mostly with LALR(1) grammars, with sometimes powerful additionals as the disambiguating rules in Yacc. In case of failure, the manual transformation of the grammar to get an LALR(1) grammar presents the following disadvantages:
- it is arduous;
- it results often in an unintelligible semantic presentation;
- it is dangerous as one may obtain the grammar of a superlanguage;
- it is reserved for specialists.

We may regret the refusal of LR(k) grammars, but we also deplore the nonacceptance of LRR grammars, of which we will recall Cohen and Culik's definition: a CFG grammar is LRR if there is a regular partition \( \delta \) of \( T^* \) such that

\[
S \xrightarrow{\star} \beta_1 A y_1 \xrightarrow{\star} \beta_1 \mu y_1, \\
S \xrightarrow{\star} \beta_2 B y_2 \xrightarrow{\star} \beta_1 \mu y_2 \text{ and } y_1 = y_2 \text{ (mod } \delta) 
\]

implying \((\beta_1, A, y_2) = (\beta_2, B, y_3)\).
The LR(k) is just a particular case for the regular partition
\[ \{u_1 \ldots u_n, w_1 T^*, \ldots, w_p T^*\}, \]
where the \( u_i \) are words included in \( T^k \) (whose
lengths are shorter than \( k \)) and the \( w_i \) are words whose lengths are equal to \( k \). The
set of LRR languages strictly includes the set of deterministic languages and is
strictly included in the set of nonambiguous languages.

The problem for the implementation of more powerful parser generators is
well-known for the LR(k) case when \( k \geq 1 \). Let us recall that the production of
LR(k) automata through Knuth’s algorithm shows to be unrealistic because of the
enormous number of states this entails. In fact, as quite rightly noticed by Pager,
the duplication of LR0 states that are generated by the algorithm gets out of
proportion in comparison with the problems to be resolved. Note, however, that
(a) the number of conflicts within the LR0 automata that are not LALR(1) is
generally very small compared to the number of states of the automaton;
(b) the examination of the clashing surface which solves an LR(k) conflict mostly
results in a very small duplication of LR0 collections in comparison with the one
obtained by the Knuth’s algorithm.

These statements tend to show that Knuth’s algorithm is not suited for non-
LALR(1) grammars, and that the solution lies in the search for a local analysis of
non-LALR(1) conflicts. From this point of view, the construction of reduced LR(1)
parsers by merging compatible states and the regeneration of conflicting areas are
two important contributions of Pager.

In order to use the strength of the LRR definition, the realistic previous endeavours
(XLR [9], RLR [11], LAR [10]) used the LR0 to build regular covers of the
right-context languages associated with each conflicting action, but partly ignored
the information which lies in the stack; they failed, for instance, as soon as the
same right-context word is introduced by two separate prefixes on two different
actions, leaving aside simple LR(1) or LR(2) grammars. In [10, 11] it is suggested
to use a bounded stack memory in order to refine the discrimination, but this proves
to be
- ineffective behind the cycles,
- costly when implemented.

For instance, the following trivial grammar

\[
S \rightarrow aTa | bTb | aUb | bUa,
\]
\[
T \rightarrow aTa | c,
\]
\[
U \rightarrow aUa | c
\]
is neither XLR nor \( R(h) \) LR0 nor LAR(h) for any \( h \).

These statements lead us quite naturally to take into consideration all that can
be done locally, and to present analysing techniques that solve the latter problems
independently of any Knuth LR(k) automaton.

The method we will present in Section 2 generates, whenever possible, a small
number of complementary tables to solve a conflicting state of the LR0 parser and
it can very likely become a powerful addition to the current parser generators, allowing the deterministic parse of LR(k) and non-LR(k) grammars in a realistic way. The discriminating power of the algorithm allows a potentially unbounded examination of the right context as well as an arbitrary reading of the stack. Of course, the reading of the stack can always be replaced by a splitting phase of the LR0 automaton. A schematic presentation of the extension is given in Figs. 1 and 2. The details of the interconnection with Yacc are listed in Appendix F.

![Fig. 1. Extension of current parser generators.](image)

We chose to present the results that can be widely understood in the first two sections, leaving a more technical part on LR(k) automata for the last section (direct matrix evaluation of LR(k) copies from the LR0 and production of LRk(V) independently of Knuth's algorithm).
1. A new characterization of LR(0) automata

With some definitions and notations, this first section and the third one will use two well-known facts, already mentioned in earlier literature [5, 19, 30, 31, 32]:

(1) An LR(k) state is perfectly defined by its nucleus ('kernel' in [5]); the Yacc implementation uses this characteristic;
(2) Knuth's algorithm is a subset construction.

Apart from improved implementation features, the main interest of a characterization on the nuclei consists in being able to consider an LR(k) automaton as a disjoint union of LRk(V)'s where V is a grammatical variable, and to define production algorithms for these LRk(V)'s independently of Knuth's algorithm. These algorithms are further based on the fact that the LRk(V)'s are the representatives of a partition on prefixes reaching the nuclei whose left-label is V in a rightmost derivation.

At the end of this section we will give a production algorithm for LR0(V) used for the method presented in the second section. The advantage of this is that the production of complementary tables can work independently of the parser generator.

1.1.

The CFG grammars described in the present remarks are supposed to be reduced and not circular, i.e., each variable is accessible, it is productive at least of ε (the empty string in T*), and none of the variables satisfy V ⇒+ V, which is a sufficient condition for ambiguity. Each CFG grammar of the form \( G = (N, T, P, S) \), where \( N \) stands for the set of nonterminals, \( T \) for the set of terminals, \( P \) for the set of productions and \( S \) for the axiom, is extended to a grammar \( G' = (N \cup \{S'\}, T \cup \{(\langle \rangle)\}, P \cup \{S' \rightarrow \langle(S)\}\}, S') \), where \( S' \), "(" and ")" are new symbols not in \( N \cup T \). "(" will be used as stack bottom and ")" for the end of input. From now on, \( N \cup T \) will designate the new set of variables, and \( P \) the new set of production rules.

\( T^*k \) designates the set of words of \( T^* \) whose length is less or equal to \( k \). \( ε \) is the empty string in \( T^* \). Firstk has its usual signification with Firstk(ε) = {ε}. +k is the concatenation limited to \( k \) characters. \( P(E) \) stands for the powerset of \( E \). T(L) stands for the transpose of the language \( L \). \( \cup \) and \( \cap \) respectively stand for union and intersection over sets.

Greek and Latin letters are used in the following way: All capital Latin letters belong to \( N \), except for \( X \) which belongs to \( N \cup T \), and small Latin letters denote elements of \( T^* \). The distinction between terminals and strings of terminals should be evident from the context. Greek letters denote elements of \( (N \cup T)^* \). Unfortunately, the limitations of our text-editing machine forced us to refrain from some traditions and we will use \( β \) and \( μ \) extensively, sometimes with indices.

\(^1\) In figures and appendices, the empty word \( ε \) is also denoted by \( ε \).
The following grammar denoted \( G_0 \):

\[
\begin{align*}
S' &\rightarrow (S), \\
S &\rightarrow S + T | T, \\
T &\rightarrow T * F | F, \\
F &\rightarrow (S) | a
\end{align*}
\]

will be used to illustrate this section.

1.2.

Within the set of dotted rules, called items of degree 0 in the literature and denoted \( RD_0 = \{ W \rightarrow \mu.\beta \mid W \rightarrow \mu \beta \in P \} \), we distinguish:

- nucleus rules \( W \rightarrow \mu.\beta \) where \( \mu \neq \varepsilon \),
- closure rules \( W \rightarrow \beta \).

\( RN_0 \) designates the set of nucleus rules.

A dotted rule \( W \rightarrow \mu.X\beta \) will be considered strong if \( X \in N \), and weak otherwise. A collection of nuclei which contains at least one strong nucleus is called a production center.

The left label of a nucleus rule \( W \rightarrow \mu.X\beta \) is \( X \).

To each variable \( X \in N \cup T \), the set of nuclei whose left label is \( X \) will be associated. This set is denoted \( RN(X) \). For instance, for \( G_0 \) we have

\[ RN(T) = \{ T \rightarrow T \ast F, S \rightarrow T, S \rightarrow S + T \}. \]

The relation \( RO \) on \( N \times N \) is defined by

\[ W \ RO \ V \ \text{iff} \ \ W \rightarrow V\beta \in P. \]

With \( RO^* \) and \( RO^+ \) we mean the usual closures. That is, for \( G_0 \) we get the following diagram:

\[ \overset{S}{\raisebox{-3pt}{$\longmapsto$}} \overset{T}{\raisebox{-3pt}{$\longmapsto$}} \overset{F}{\raisebox{-3pt}{$\longmapsto$}} \]

1.3.

We denote by \( d-\text{Cl}(Q, V, q_0, \{ q_f \}) \) the DFA that is derived from the NDFA \((Q, V, Q', q_0, \{ q_f \})\) where, for \( a \in V \),

\[
\begin{align*}
q_j &\in d'(q_i, a) \ \text{iff} \ q_j \in d(q_i, a), \\
q_j &\in d'(q_i, \varepsilon) \ \text{iff} \ q_j \ Cl \ q_j.
\end{align*}
\]

Here, \( q_0 \) stands for the initial state, \( q_f \) are the final states and the DFA is obtained by the classical algorithm (subset construction):

\[
\text{initial collection: unmarked closure}(q_0)
\text{while there is an unmarked collection}
\text{mark } Ci
\text{for each } X \text{ belonging to } V
\text{calculate } Cj = \text{closure}(d(Ci, X))
\text{endfor}
\text{endwhile.}\]
Here, \( \text{closure}(qu) = \{qv | qu Cl^*qv \} \). The final collections are the collections which contain at least one \( qfi \).

When the NDFA is \( \varepsilon \)-free (no \( Cl \)), we note \( d-\emptyset(Q, V, q0, \{qfi\}) \).

1.4.

In this paper we will call the following automaton a Knuth-L \( Rk(G) \) automaton:

\[
ds-Kcl[ITEMS_{k}, V, (S' \rightarrow \langle S \rangle), \{\varepsilon\}], (S' \rightarrow \langle S \rangle), \{\varepsilon\}]
\]

where \( ITEMS_{k} = \{(W \rightarrow \mu_{\beta}, u)\} \) with \( u \in T^*_k \) and \( W \rightarrow \mu_{\beta} \in \text{RD}_0 \);

\[
(W \rightarrow \mu_{Z\beta}, u) Kcl (Z \rightarrow \alpha, v) \quad \exists \in N \text{ and } v \in \text{First}_k(\beta u);
\]

Further, \( ds[(W \rightarrow \mu_{X\beta}, u), X] = (W \rightarrow \mu_{X\beta}, u) \). (For \( k = 0 \) take away the second component.) For \( k = 0 \), a characterization of the nuclei can easily be deduced from this definition, as shown in the following lemma.

**Lemma 1.1.** (a) Knuth-L \( R0(G) \) is identified with

\[
d0-\emptyset(\text{RN}_0, N \cup T, (S' \rightarrow \langle S \rangle), (S' \rightarrow \langle S \rangle)).
\]

where \( d0 \) is the union of two functions \( ds0 \) and \( dg0 \)

\[
ds0[(W \rightarrow \mu_{X\beta}, X)] = (W \rightarrow \mu_{X\beta}),
\]

\[
dg0[(W \rightarrow \mu_{Y\beta}, X)] = [(Z \rightarrow X1.\alpha) | Y \text{ RO}^* Z].
\]

(b) For any collection \( Ci \) of \( d0-\emptyset(\text{RN}_0, N \cup T, (S' \rightarrow \langle S \rangle), (S' \rightarrow \langle S \rangle)). \) we have

(i) \( ds0(Ci, X) \cap dg0(Ci, X) = \emptyset \),

(ii) left-label\( (Ci) \) is perfectly determined as the common value of the left-label of all the nuclei of \( ds0(Ci, X) \) and \( dg0(Ci, X) \).

Before we give a quick proof of these two propositions, an example will illustrate the transition mechanism: For \( G0 \) the following scheme will demonstrate the RO relation:

\[
\bigcup S \rightarrow T \rightarrow F
\]

\[
d0[(S \rightarrow \langle S \rangle), T] = ds0[(S \rightarrow \langle S \rangle), T] \cup dg0[(S \rightarrow \langle S \rangle), T]
\]

\[
\emptyset \cup \{(S \rightarrow T), (T \rightarrow T^*F)\}
\]

For \( dg0 \) we looked for all productions beginning with \( T \) and whose left-nonterminal \( W \) satisfies \( S \text{ RO}^* W \). The interest of decomposing \( d0 \) into \( ds0 \) and \( dg0 \) will become clear when studying LR\( (k) \) automata in Section 3. Imaginatively, \( ds0 \) 'shifts the rules' and \( dg0 \) 'generates a new set of dotted rules'.

**Proof of Lemma 1.1.** (a): We suppress the superfluous transitions in the Knuth
closure to define a new closure on the nuclei onto closure rules. 

\[ \text{Clo}(W \to \mu.X\beta) = \begin{cases} \emptyset & \text{if } X \in T, \\ \{(Z \to \alpha)/X \ R^* Z\} & \text{otherwise.} \end{cases} \]

It is easy to verify that \( \text{Clo}(C_i) = \text{Knuth-Closure}(C_i) \) for all collections of nuclei.

d\( s_0 \) is the traditional Knuth transition function restricted to \( \text{RN}_0 \).

By the well-known technique for obtaining an \( \varepsilon \)-free automaton, we get from

\[ \text{NDFA}[R^0, \{(S' \to \langle .S \rangle), (S' \to \langle S \rangle)\}] \]

the automaton

\[ \text{NDFA}[R^0, \{\diamond_0, (S' \to \langle .S \rangle), (S' \to \langle S \rangle)\}] \]

and, finally, the DFA \( \diamond_0 - \varepsilon \).

(b) (i): An immediate consequence of the definition of \( ds_0 \) and \( dg_0 \) which entails that a rule shifted by \( ds_0 \) has a dot positioned at \( \dagger \) on position 2 whereas rules generated by \( dg_0 \) have their dot on position 1.

(b) (ii): Also an immediate consequence of the definition of transition functions. \( \Box \)

1.5.

For \( n \) nonempty sets \( A_i \), we call viable intersections or viable minterms, the partition's elements for the equivalence on \( \bigcup A_i \)

\[ a = b \iff \left[ \forall i \in [1, n] (a \in A_i \iff b \in A_i) \right]. \]

If we associate, with each nonempty subset \( I \) of \( \{1, \ldots, n\} \), \( I = \{i_1, \ldots, i_k\} \), called indicator, the set defined as follows: \( R(I) = \bigcap A' \) where

\[ A' = \begin{cases} A_i & \text{if } i \in \{i_1, \ldots, i_k\}, \\ \bigcup A_j - A_i & \text{otherwise (} j \in [1, n] \). \end{cases} \]

then we immediately get that \( C \) is a 'viable intersection' iff \( \exists I/C = R(I) \neq \emptyset \). The indicators \( I \) for which \( R(I) \neq \emptyset \) will be called viable indicators.

This yields a useful lemma which we will need further on.

**Lemma 1.2.** On \( d_0(Q, V, q_0, \{q_f\}) \) there will be as many collections as 'viable intersections' on \( \bigcup Kq_0q_i \) where \( \{q_i\} \) designates the set of accessible states from \( q_0 \) on the underlying NDFA and \( Kq_0q_i = \{w \in V^* | (q_0, w) \vdash^* (q_i, \varepsilon)\} \).

**Proof.** If \( C_j = \{q_1l_1, \ldots, q_kl_k\} \) is a collection on the DFA, we get, by an immediate induction,

\[ (C_0, w) \vdash^t (C_j, \varepsilon) \iff (q_0, w) \vdash^t (q_1l, \varepsilon) \text{ for all } q_1l \in C_j \]

and then

\[ K C_0 C_j = \{w \in V^* | w \in Kq_0q_{i\mu} \text{ and } w \notin Kq_0q_{i\nu}\}, \]
where \( i u \subseteq \{i_1, \ldots, i_k\} \) and \( v \not\subseteq \{i_1, \ldots, i_k\} \); or \( KC0Cj = \bigcap K'0q0qj \) where

\[
K'0q0qj = \begin{cases} 
Kq0qj & \text{if } i \in \{i_1, \ldots, i_k\}, \\
Kq0qj - Kq0qj & \text{otherwise.}
\end{cases}
\]

At last, the \( KC0Cj \) are pairwise disjoint and \( \bigcup KC0Cj = \bigcup Kq0qj \).

We conclude that the \( KC0Ci \) are the 'viable intersections' on \( \bigcup Kq0qj \) and the \( Ci \) are in bijection with them. The \( Ci \) can be seen as the viable indicators on \( \bigcup Kq0qj \). □

1.6.

For each nucleus rule \( ni : W \rightarrow \mu X.\beta \) we denote by \( Kni \) the regular set of all prefixes reaching \( ni \) in a rightmost derivation:

\[
K(W \rightarrow \mu X.\beta) = \{\alpha \mu X | S' \xrightarrow{\mu} \alpha Wx \xrightarrow{\mu} \alpha \mu X\beta x\}.
\]

We note by \( KN(X) \) the set \( \bigcup Kni \) where \( ni \in RN(X) \). On the set \( RN'0 \) of nuclei augmented with a new rule \( S' \rightarrow @. \), we define the function \( dr0 \) by

\[
dr0[(W \rightarrow \mu X.\beta), X] = \begin{cases} 
(W \rightarrow \mu X.\beta) & \text{if } \mu \neq \varepsilon, \\
\{(Z \rightarrow \mu 1.A\beta 1) | A R^* \varepsilon \} & \text{otherwise}
\end{cases}
\]

for all nuclei of the augmented grammar except \( S' \rightarrow \langle S \rangle \), for which \( dr0 \) is defined by

\[
dr0[(S' \rightarrow \langle S \rangle), \langle \rangle] = (S' \rightarrow @.) \quad \text{(the new dotted rule)},
\]

\[
dr0[(S' \rightarrow @.), X] = \emptyset.
\]

We obtain the transpose of \( K(W \rightarrow \mu.\beta) \) on the DFA

\[
dr0-\emptyset(RN'0, N \cup T, (W \rightarrow \mu.\beta), (S' \rightarrow @.))
\]

as follows:

\[
T(K(W \rightarrow \mu.\beta)) = dr0-\emptyset(RN'0, N \cup T, (W \rightarrow \mu.\beta), (S' \rightarrow @.))
\]

For instance, for \( G0 \), the transpose of \( K(S \rightarrow S + T) \) is shown in Fig. 3.

1.7.

If we denote all the states of the LR0 whose left-label is \( V \) by \( LR0(V) \), then, from Lemma 1.2, we can deduce the next lemma.

**Lemma 1.3.** \( LR0(V) \) is in bijection with the partition's elements of the equivalence \( \equiv_0 \) defined on \( KN(V) \) by

\[
\mu_1 \equiv_0 \mu_2 \iff [\forall i \in [1, n](\mu_1 \in Kni \iff \mu_2 \in Kni)].
\]
Here, the ni are elements of RN(V) and n = Card(RN(V)).
LR0(V) are determined by the indicators of the 'viable intersections' on KN(V).

This lemma is the justification of the Aut0pref(V) automaton described at the end of this section, which generates LR0(V).

Proof of Lemma 1.3. As KN(V) \neq KN(V') if V \neq V', we deduce the lemma from Lemmas 1.1 and 1.2 and from the viable-prefix property. (The nuclei ni stand for the qi, and Kni for \langle Kq0\rangle in Lemma 1.2.)  

1.8.

The actions of a nucleus W \rightarrow \mu. X\beta are

- **shift a** if X = a, or (X \in N and X \ RO* Y and Y \rightarrow a\mu 1 \in P);
- **reduce(Z \rightarrow \epsilon)** if X \ RO* Z and (Z \rightarrow \epsilon) \in P;
- **reduce(W \rightarrow \mu)** if X\beta = \epsilon.

The actions of a collection of nuclei are the actions of its nuclei.

A collection of nuclei is called *inadequate* if the type is shift-reduce or reduce for at least two reduces, and it is called *adequate* otherwise.

1.9. Algorithm 1: determination of LR0(V)

1.9.1. Informal description

Lemma 1.3 naturally yields a definition of an algorithm for determining the 'viable intersections' on KN(V). Here we use the dr0 function defined in Section 1.6, as
well as the well-known technique of deterministically merging $n$ automata. An example will illustrate this: For $G_0$,

$$\text{RN}(S) = \{(S' \rightarrow \langle S \rangle), (S \rightarrow S + T), (F \rightarrow \langle S \rangle)\},$$

and we get the initial collection with three couples

$$S' \rightarrow \langle S \rangle, 1$$

$$S \rightarrow S + T, 2$$

$$F \rightarrow \langle S \rangle, 3$$

We mark all the nuclei with an index. Now, if we make the subset construction simultaneously for the three rules by use of $drO$ we get the result in Fig. 4. We stop as soon as the collection has a unique set of indices for each rule in it. (Then we say that the collection is equidistributed.) In fact we can even stop as soon as a transition from a collection concerns a unique set of indices for each rule sharing the transition. It was done this way in Fig. 4. For one equidistributed collection

![Fig. 4.](image)

we call the set of indices concerned indicator. Further, we define INDICATOR as a mapping which associates, with the initial collection of nuclei, all collections represented by the indicators. We anticipate the proof of our algorithm and say that $\text{LR}_0(S)$ is the set of indicators $[\text{INDICATOR}(\text{RN}(S))]$. $\text{LR}_0(S) = \{1, 2\}, \{3, 2\}$ or

<table>
<thead>
<tr>
<th>collection no. 1</th>
<th>collection no. 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S' \rightarrow \langle S \rangle$</td>
<td>$S \rightarrow S + T$</td>
</tr>
<tr>
<td>$S \rightarrow S + T$</td>
<td>$F \rightarrow \langle S \rangle$</td>
</tr>
</tbody>
</table>

The same work on $\text{LR}_0(T)$ results in Fig. 5 and the collections

<table>
<thead>
<tr>
<th>collection no. 3</th>
<th>collection no. 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S \rightarrow S + T. T$</td>
<td>$S \rightarrow T^* F$</td>
</tr>
<tr>
<td>$T \rightarrow T^* F$</td>
<td>$S \rightarrow T. T$</td>
</tr>
</tbody>
</table>

![Fig. 5.](image)
For LR0(F) we find the result in Fig. 6 and the collections

\[ T \rightarrow T^*F. \quad T \rightarrow F. \]

Here ends the determination of LR0(\(G0\)) as we have just one element for the other RN(\(Vi\)) which constitutes on its own the unique state of LR0(\(Vi\)).

\[
\begin{align*}
\text{LR0}(+) & : S \rightarrow S+.T \quad \text{col 7} \\
\text{LR0}(*) & : T \rightarrow T^*.F \quad \text{col 8} \\
\text{LR0}(() & : F \rightarrow (.)S) \quad \text{col 9} \\
\text{LR0}(()) & : F \rightarrow (S). \quad \text{col 10} \\
\text{LR0}(a) & : F \rightarrow a. \quad \text{col 11} \\
\text{LR0}(\langle) & : S' \rightarrow \langle.$S$\rangle \quad \text{col 12} \\
\text{LR0}(\rangle) & : S' \rightarrow \langle.$S$\rangle. \quad \text{col 13.}
\end{align*}
\]

This construction will, of course, apply to any set of initial nucleus rules, and we will now elucidate this statement by an algorithm.

1.9.2. Algorithm 1

Algorithm 1

Input: \(n\) nuclei numbered from 1 through \(n\).
Output: a partition of prefixes reaching them by a rightmost derivation.
Objects: Cartesian product--\(RN'0 \times P\{1 \ldots n\}\)
For LR0(\(V\)) a list collecting the indicators
Initial collection

\[ \{(p1, 1) \ldots (pn, n)\} = C0 \quad \text{unmarked} \]
empty list $L$. 

while there is an unmarked collection $C_i$ 
mark it 
for each $X \in N \cup T$
calculate $C_j = dr_0(C_i, X)$ 
if $C_j$ is equidistributed 
$C_j$ is final and marked 
if indicator($C_j$) $\notin L$

$L = L \cup$ indicator($C_j$) 
endif 
endif 
endfor 
endwhile 
write $L$ (for LR0($V$))

We call this automaton Aut0pref-restricted($C_i$), where $C_i$ is the collection of examined nuclei.

If we restrict the definition of equidistributed collections to collections of the form $S' \rightarrow \@, \{i_1, \ldots, i_k\}$, we get Aut0pref($C_i$).

1.9.3. Proof of Algorithm 1

LR0($V$) = INDICATOR(Aut0pref-restricted(RN($V$))) = INDICATOR(Aut0pref(RN($V$))).

Proof. For Aut0pref, with respect to the indicator $I = \{i_1, \ldots, i_k\}$, we have one single final state: $C/f : S' \rightarrow \@, I$. From Section 1.7 and from the deterministic merge of $n$ automata, we find $T(KC0Cf) = \bigcap K'ni$ with

$$K'ni = \begin{cases} Kni & \text{for } i \in \{i_1, \ldots, i_k\}, \\ \bigcup Knj - Kni & \text{for } i \notin \{i_1, \ldots, i_k\}. \end{cases}$$

Hence, $I$ is the indicator of a 'viable intersection' on KN($V$) and, from Lemma 1.3 it is positively an element of LR0($V$).

For all final collections $C/fI$, we have that $\bigcup T(KC0CfI) = KN(V)$.

At last, from our conditions on the grammars and from the technique used, it is easily seen that an equidistributed collection of Aut0pref-restricted, whose indicator is $I$, is coaccessible of a unique $S' \rightarrow \@, I$. Consequently, the indicators of Aut0pref and of Aut0pref-restricted coincide. \(\square\)

2. LRR parsing technique

Preliminary remark. For Algorithm 2, for an indicator $I$ we extract, out of Aut0pref(RN($V$)), a nondeterministic subautomaton in the following way: If $Si$
denotes an article of Aut0pref(RN(V)), then keep on the automaton all Si's coaccessible of S' → @, I, and make some obvious minimization. After renumbering, dpref is the transition function defined on this new set: Sj ∈ dpref(Si) iff Sj is a successor of Si on Aut0pref for some v. The transition component is obvious (left-label of Si).

This trick spares $d_0^{-1}$ thus allowing independence in the production of complementary tables from the parser generation.

Of course, with this bottom-up concept, we can very easily deduce an obvious strategy to simultaneously build the LR0 automaton and collect the LALR(1) context.

2.1. Informal description of Algorithm 2

The algorithm will examine the n conflicting actions of an inadequate LR0 state. By simulating all the rightmost derivations that can reach them, it produces a finite-state automaton for the right context. Should this automaton prove insufficient to discriminate the conflicting actions, a 'Reverse' function constructed upon the articles provides the possibility of building, for each impure collection, an automaton on the stack alphabet in order to try to discriminate the remaining conflicting actions by a bottom-up reading of the stack.

It would, of course, always be possible to reach a final 'pure-look' solution via a splitting phase of the LR0 automaton. The construction of regular covers for the right context of the actions uses a nonterminal cover (to be chosen by the user—in Section 2.8 we will offer two solutions) on which we impose First1(RCover(I)) = First1(I) in order to converse the valid prefix property and to obtain the LALR(1) context on the initial collection.

Finally, the method is also characterized by its minimization techniques which enable us to drastically reduce the size of the automata produced, as will be explained further on.

To illustrate our subject, we will give three examples of output lists for three grammars using verbose option 1:

(1) The first output list on a short grammar reflects the power of the algorithm compared with earlier attempts that have been limited to looks, and gives an example of recognition of non-LR(1) grammars.

(2) The second one illustrates the resolution of LR(k) conflicts for $k \geq 1$ for the LR(2) part of the 'natural' Yacc grammar.

(3) The third example will show the power of the minimization techniques. Then, we continue with illustrating the description of the algorithm with verbose option 2:

(4) The look collections of the Yacc example with the internal Reverse function will be examined.

(5) The collections of a stack automaton from the first example will be shown. Other examples can be found in Appendix E.

Example (1)
For the language
\{aa"ba" b, ba"ba" a, aa"ba" a, ba"ba" b},
which is nondeterministic (non-LR(k)) and has as one of its grammars

(0) \( S' \to \langle \langle S \rangle \rangle \)
(1) \( S \to aTb \)
(2) \( S \to bTa \)
(3) \( S \to aUa \)
(4) \( S \to bUb \)
(5) \( T \to aTaa \)
(6) \( T \to b \)
(7) \( U \to aUa \)
(8) \( U \to b, \)

the LR0 state

\[ U \to b. \]
\[ T \to b. \]

is neither LR(k) nor XLR nor \( R(h) \)LR0 nor LAR(h). During a first phase, the algorithm produces a four-state look automaton, reduced to three states by the minimizer; then, for the two impure collections, two one-state stack automata are

LRO variable b no: 003

\[
\text{T } \to \text{ b } \\
\text{U } \to \text{ b } \\
\text{shift} \\
\text{reduce } 6 \ 8 \\
\text{INADEQUATE} \\
\text{slr look reduce 006: a b} \\
\text{slr look reduce 008: a b} \\
\text{ACTIONS CODE} \\
\text{red 6 0} \\
\text{red 8 1} \\
\text{NOT LALR(1)} \\
\text{Articles : 41} \\
\text{Collections : 4} \\
\text{Space used : 205 bytes} \\
\text{IMPURE LOOK} \\
\text{STACK AUTOMATON NO S00 (LOOK COLLECTION A001)} \\
\text{Articles : 4} \\
\text{Collections : 1} \\
\text{Space used : 12 bytes} \\
\text{PURE STACK} \\
\text{STACK AUTOMATON NO S01 (LOOK COLLECTION A002)} \\
\text{Articles : 4} \\
\text{Collections : 1} \\
\text{Space used : 12 bytes} \\
\text{PURE STACK} \\
\text{MINIMIZATION LOOK \& STACK AUTOMATA type 2} \\
** Ax stands for action x \& Sx for Stack Automaton x ** \\
** Refer to the codes \text{**} \\
\text{S T U < a b >} \\
\text{< >} \\
\text{Look} \\
\text{LOC01} 2 S1 \\
\text{LOC02} 3 A1 S0 \\
\text{LOC03} 2 S1 A1 \\
\text{Stack} \\
\text{S0C01} A1 A0 \\
\text{Stack} \\
\text{S1C01} A0 A1 \\
\text{Fig. 7.}
built up in order to discriminate the actions. This is the situation summarized by verbose option no. 1 in Fig. 7. It can be easily noticed that right-context articles are implemented on 5 bytes and the stack articles on 3 bytes.

Example (2)

One of the nice features of Yacc is that the grammar describing it in the most natural way is not a LALR(1) but a LR(2) grammar. The part of the grammar posing problems is the following:

Grammar rules:

(0) $S' \rightarrow \langle\text{SPEC}\rangle$

(1) SPEC $\rightarrow$ RULES TAIL

(2) TAIL $\rightarrow$ mark

(3) TAIL $\rightarrow$

(4) RULES $\rightarrow$ id; RBODY PREC

(5) RULES $\rightarrow$ RULES RULE

(6) RULE $\rightarrow$ id; RBODY PREC

(7) RULE $\rightarrow$ RBODY PREC

Our algorithm gives a direct solution by means of a two-state look automaton for the three conflicting LRO states. As these solutions are similar for the three states, we extract the resolution of one state in Fig. 8.

LRO variable RBODY no: 000
RULE $\rightarrow$ # RBODY _ PREC
RBODY $\rightarrow$ RBODY _ id
RBODY $\rightarrow$ RBODY _ ACT
shift prec id {
reduce 15
INADEQUATE
slr look reduce 015: mark id # >>
ACTIONS CODE:
shift 0
red 15 1
NOT LALR(1)
Articles : 29
Collections : 2
Space used : 145 bytes

PURE LOOK
MINIMIZATION LOOK AUTOMATON type 2
** Ax stands for action x **
** Refer to the codes **

Look
LOC01 A1 2 A1 A0 A0 A1
LOC02 A0 A0 A1 A0 A0 A0 A0

Fig. 8.
Example (3)

Resuming Cohen-Culik’s example on the grammar given in Appendix D, the solution concerning one non-LR(k) conflict is given in Fig. 9. In this bad case one can easily notice how essential the minimization techniques (explained in Section 2.9) are. An automaton consisting of 125 collections and needing 13K is transformed into one of the two collections appropriate to solve the problem.

```
ANALYSIS FOR VARIABLES ID
LR0 variable ID no: 003
F -> ID _
SG -> ID _
ID -> ID _ L
shift a b
reduce 17 26
INADEQUATE
slr look reduce 017: ; = then + - * ) >>
slr look reduce 026: ; = then + - * ) >>
```

**Fig. 9.**

**Example (4)**

Verbose option no. 2 gives us (in Fig. 10) a human-readable description of the collections in Example (2) as well as the internal Reverse so that the closure mechanism within a collection can be understood. The meaning of the fields will be given in the next subsection.

**Example (5)**

A description of a stack automaton from the first example is given in Fig. 11.

2.2. Object description

Fivefolds, denoted Ai, which exist in two types (type-article) (1): context and (2): prefix, will both be discussed below.
LOOK COLLECTION NO: A000
no context or "var" read tv f pref ac internal Reverse
00 S' -> << SPEC --> dir 1 S5 1 2,
01 SPEC -> RULES TAIL dir 0 S5 1 12,
02 RULES -> RULES TAIL dir 0 S5 1 1,
03 RULES -> RULES RULE dir 0 S5 1 12,
04 RULES -> RULES RULE dir 1 S4 1 14,
05 RULE -> # RBODY PREC _ dir 1 S0 1 11,
06 RBODY -> RBODY _ id dir 1 S1 0
07 ACT -> _ { } dir 0 S2 0
08 PREC -> _ prec id dir 0 S0 0
09 PREC -> _ prec id ACT dir 0 S0 0
10 PREC -> _ prec ; dir 0 S0 0
11 PREC -> _ dir 0 S0 1
12 ~ RULES 0 S4 1 4,
13 ~ RBODY 0 S0 1 5,
14 ~ # 0 S3 1 13,

shift { prec
red 15 mark # >>
TRANSITIONS:

Fig. 10.

Fig. 10.

where the component Si is given in the table
Si VALUE dpref
S0 RULE -> # RBODY _ PREC S3
S1 RBODY -> RBODY _ id S3
S2 RBODY -> RBODY _ ACT S3
S3 RULE -> # _ RBODY PREC S4
S4 RULES -> RULES RULE S5
S5 S' -> << _ SPEC >> S6
S6 S' -> @

2.2.1. Context

We have the following five objects: component-context, type-reading, flag-objective, component-prefix, no action. For instance, there is the excerpt of example (4) in Section 2.1 where the component Si has been translated.
STACK AUTOMATON NO S00 (LOOK COLLECTION A001)
STACK COLLECTION NO: B000
no A, ... action
000 A001,00 0
001 A001,11 0
002 A001,01 1
003 A001,09 1

red 6 b
red 8 a
TRANSITIONS:
FINAL

Articles : 4
Collections : 1
Space used : 12 bytes

Fig. 11.

<table>
<thead>
<tr>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
<th>C5</th>
</tr>
</thead>
<tbody>
<tr>
<td>RULES→RULES RULE_</td>
<td>dir 1</td>
<td>RULES→RULES RULE 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPEC→RULES TAIL_</td>
<td>dir 0</td>
<td>$S'\rightarrow\langle _SPEC \rangle$</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>RULES→RULES RULE_</td>
<td>id T</td>
<td>$S'\rightarrow\langle _SPEC \rangle$</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Component-context is a dotted rule produced automatically by the algorithm.

Type-reading is either dir (direct) or a state $q$ of the examined cover. (Our cover is read on two fields, 'read' and 'tv'. It will be explained further on.)

Flag-objective is a bit, “on” while reading the context of the prefix component and “off” otherwise.

Component-prefix (stack memory) is obtained through an external function deduced from the 0-prefix automaton that was used to determine the LR0(V) collections (cf. Aut0pref(RN(V)) from Section 1), or by remounting the LR0 on the nuclei. The dot is to trace the viable prefixes. We could content ourselves with dr0 but then the covers would be broader and the discrimination prefix less precise. From now on, we will call this function dpref and denote by $Si$ the component prefixes. The transition component is left aside because it is self-evident (left-label of $Si$). (Fig. 10 presents one table of $Si$'s with transitions related to example (4).)

The field labelled $ac$ is a code for the action concerned.

A context article is then implemented in 5 or 7 bytes, depending on the size of the accepted grammars and the skills of the programming staff.

For a more convenient description of the algorithm (given in Appendix A), we will distinguish several types of context articles. The context component is said to be dead if the dot is at the extreme right and the reading mode is “dir”, e.g.,

RULES→RULES RULE_ dir 1 RULES→RULES RULE 1

The article is called completely dead if the context component is dead and the flag is set at 1, e.g.,

RULES→RULES RULE_ dir 1 RULES→RULES RULE 1
and _half dead_ if the context component is dead and the flag is set at zero, e.g.,

\[
\text{SPEC} \to \text{RULES TAIL} \quad \text{dir} \ 0 \ S' \to \langle \_\text{SPEC} \rangle \ 0
\]

The article is said to be _alive_ otherwise; e.g.,

\[
\text{RULES} \to \text{RULES..RULE} \quad \text{id} \ 0 \ S' \to \langle \_\text{SPEC} \rangle \ 1
\]

These three possibilities define the _type-context_ of a context article.

### 2.2.2. Prefix

These articles can be found at the end of the list in Fig. 10 and are distinguished by `~` on their first component. For the objects, the three first components are irrelevant; the fourth is a component prefix Si, and the fifth is the number of the action; e.g., the excerpt of example (4) of Section 2.1 translating Si:

\[
\text{~ #} \ 0 \ \text{rule} \to \# \ _\text{RBODY PREC} \ 1
\]

When implementing our method we use the second component, here `#`, to store the stack transitions so that only _one_ economical Reverse function is needed to build the stack automata. As for the context articles, we will distinguish two types: if the component prefix has the form \(W \to X.\beta\), we will say that the article is _unstable_, e.g.,

\[
\text{~ #} \ 0 \ \text{RULE} \to \# \ _\text{RBODY PREC} \ 1
\]

Otherwise we call it _stable_, e.g.,

\[
\text{~ RBODY} \ 0 \ \text{RULE} \to \# \ _\text{RBODY ..PREC} \ 1
\]

These two possibilities define the _type-prefix_ of a prefix article.

### 2.3. Production of the right-context automaton

This automaton is the DFA we get through

\[
dt-(\text{Clc} \cup \text{Clp})[A_i, T; \{\text{initials}\}]
\]

where the mono-action transitions are directed on the final states representing the actions, as illustrated in the preceding examples. For impure collections see Section 2.5. For the definition of the initials see Section 2.4. For an algorithmic description of Clc \(\cup\) Clp see Appendix A.

dt is a transition function on the \(A_i\)'s, and Clc and Clp are two closure functions as will be explained below. Furthermore, during this subset construction, two functions, crc and drp, are constructed as follows:

- (a) \(Ai \in \text{crc}(Aj)\) iff \(dt(Ai, a) = Aj\) for some \(a \in T\) or \((Aj \in \text{Clc}(Ai))\)
- (b) \(Ai \in \text{drp}(Aj, v)\) iff \(Aj \in \text{Clp}(Ai)\) and left-label(Si) = \(v\), where \(Si\) is the prefix-component of \(Ai\) (type-prefix) and \(v\) its left-label.

As explained above, the storing of \(v\) for drp happens on one of the available components of the prefix articles and then crc and drp are implemented as being
one and only one \( \epsilon \)-function \textit{Reverse}. On the preceding outputs, internal \textit{Reverse} is given for each collection context.

A formal description of the way Clc, Clp and dt work on the fivefolds will be presented below. When the value of a component is irrelevant this will be marked by a full stop.

### 2.3.1. Clc

Clc works in the four following cases:

(a) \((A \rightarrow \mu. I \beta, \text{dir, }., ., .) \text{ Clc } (A \rightarrow \mu. I \beta, \text{dir, }., ., .) \text{ if } I \xrightarrow{\ast} \epsilon;\)

(b) \((Z \rightarrow \mu. I \beta, \text{dir, }., ., .) \text{ Clc } (Z \rightarrow \mu. I \beta, q^i, \text{dir, }., ., .)\)

where \(q^i\) is the initial collection of the regular cover of \(I\);

(c) \((Z \rightarrow \mu. I \beta, q^i, \text{dir, }., ., .) \text{ Clc } (Z \rightarrow \mu. I \beta, \text{dir, }., ., .)\)

where \(q^i\) is a final state of the regular cover of \(I\);

(d) (prefix-free ascent)

\((T \rightarrow \text{something, dir, }0, W \rightarrow \beta. T \mu, .) \text{ Clc } (W \rightarrow \beta. T \mu, \text{dir, }1, W \rightarrow \beta. T \mu, .)\)

and, for all \(Y \rightarrow Z \mu 1/T \text{ RO* } Y,\)

\((Z \rightarrow \text{something, dir, }0, W \rightarrow \beta. T \mu, .) \text{ Clc } (Y \rightarrow Z \mu 1, \text{dir, }0, W \rightarrow \beta. T \mu, .)\)

These two cases are not exclusive.

### 2.3.2. Clp

The point here is to go from an entirely read prefix-free ascent to all the possible following ones while storing the stack transition on the second component. The result will be a succession of prefix articles which we will mark with a \(\sim\) on the first component. This mechanism can be divided into two phases:

\textbf{Phase 1:}

\((W \rightarrow X_1--X_n T \beta, \text{dir, }1, W \rightarrow X_1--X_n T \beta, .) \text{ Clp } (^*, X_n, 0, W \rightarrow X_1--X_n T \beta, .)\)

\((^*, X_n, 0, W \rightarrow X_1--X_n T \beta, .) \text{ Clp } (^*, X_n - 1, 0, W \rightarrow X_1--X_n T \beta, .)\)

etcetera.

\textbf{Phase 2:} When the dot is in position 1, it will be the position where the LR0 automaton through \(d0-1\) or our prefix automaton through dpref is used to select, among all possible prefix-free ascents, the ones that are permitted. For all \(W_1 \rightarrow \mu_1. T_1 \beta_1 \in \text{dpref}(W \rightarrow X_1.X_2--X_n T \beta),\)

(a) if \(W = T_1, \text{ then}\)

\(\left(^*, X_1, 0, W \rightarrow X_1.X_2--X_n T \beta, .\right) \text{ Clp } (W_1 \rightarrow \mu_1 T_1 \beta_1, \text{dir, }1, W_1 \rightarrow \mu_1 T_1 \beta_1, .)\)
(b) for all $Z \to W\alpha \mid T1 \text{RO}^* Z$,

$\text{(}, X1.0, W \to X1.X2--XnT\beta, .) \text{Clp} \ (Z \to W.\alpha, \text{dir}, 0, W1 \to \mu 1.T1\beta1, .)$.  

These two cases are not exclusive again.

2.3.3. $dt$

The terminals are read in a natural way:

(a) $dt[(Z \to \mu 1.a\beta 1, \text{dir}, ., ., .), a] = (Z \to \mu 1.a\beta 1, \text{dir}, ., ., .)$

and, for the progress on the cover of the nonterminals,

(b) for all $qj a$-successors of $qi$ on the regular cover of $I$,

$dt[(Z \to \mu 1.I\beta 1, qi, ., ., .), a] = \{(Z \to \mu 1.I\beta 1, qi, ., ., .)\}$.

In Appendix A we will show the algorithm unifying Clc and Clp.

2.4. The initial articles

For each nucleus of the LR0 collection, its actions will now be detailed and initial articles are created through the following procedure.

Procedure

for each nucleus $ni W \to \mu.X\beta$ of the collection{

switch($X\beta$){

case $X\beta = \varepsilon$:

create($W \to \mu., \text{dir}, 1, ni, \text{reduce}(W \to \mu.)$);

break;

case $X$ terminal:

create($W \to \mu.X\beta, \text{dir}, 1, ni, \text{shift}$);

break;

default:

for each $Z/X \text{RO}^* Z$

if($Z \to \varepsilon \in P$

create($Z \to \varepsilon., \text{dir}, 0, ni, \text{reduce}(Z \to \varepsilon.)$);

for each $Z \to a\delta \in P$

create(($Z \to .a\delta, \text{dir}, 0, ni, \text{shift}$))

endfor

endfor
}

endswitch

endfor.

The actions are, of course, coded as shown in the examples: In the first example of Section 2.1 the two nuclei are of type 'reduce' and therefore we have

$T \to b\_, \text{dir}, 1, T \to b\_, 0$

$U \to b\_, \text{dir}, 1, U \to b\_, 1$
In the second example in Section 2.1

\[
\text{RULE} \rightarrow \# \text{RBODY\_PREC}
\]

produces shift actions

\[
\begin{align*}
08 & \quad \text{PREC} \rightarrow \_\text{prec id} \quad \text{dir} \quad 0 \quad S0 \quad 0 \\
09 & \quad \text{PREC} \rightarrow \_\text{prec id ACT} \quad \text{dir} \quad 0 \quad S0 \quad 0 \\
10 & \quad \text{PREC} \rightarrow \_\text{prec;} \quad \text{dir} \quad 0 \quad S0 \quad 0
\end{align*}
\]

and a reduce action (\text{PREC} \rightarrow \varepsilon.)

\[
11 \quad \text{PREC} \rightarrow \_ \quad \text{dir} \quad 0 \quad S0 \quad 1
\]

\text{RBODY} \rightarrow \text{RBODY\_id} \text{ produces a shift action}

\[
06 \quad \text{RBODY} \rightarrow \text{RBODY\_id} \quad \text{dir} \quad 1 \quad S1 \quad 0
\]

and \text{RBODY} \rightarrow \text{RBODY\_ACT} \text{ produces a shift action}

\[
07 \quad \text{ACT} \rightarrow \_\{\} \quad \text{dir} \quad 0 \quad S2 \quad 0
\]

2.5. Impure collections

A collection is \textit{impure} if it contains at least two articles:

\[
\begin{align*}
S' \rightarrow \langle(S.)\rangle, \text{dir}, 1, & \quad S' \rightarrow \langle(S.)\rangle, \text{action} \ i \\
S' \rightarrow \langle(S.)\rangle, \text{dir}, 1, & \quad S' \rightarrow \langle(S.)\rangle, \text{action} \ j
\end{align*}
\]

where \(i \neq j\). These articles are called \textit{final}.

2.6. Production of a stack automaton on impure collections

2.6.1. Objects

Denoted \(B_i\), objects are pairs \((i, \text{action code})\) where \(i\) designates the number of an article \(A_i\) and the action code is the one of \(A_i\). Once implemented they will take a maximum of three bytes.

2.6.2. Algorithm

With all final articles \(A_{ik}\) of the impure collection \(\cup fi\), a set of pairs is associated \((i_k, \text{action code of } A_{ik})\). The set consisting of these couples is denoted \(\text{INIPREF}(Ci)\).

The stack automaton is \(\text{drp-crc}(Bi, N \cup T, \text{INIPREF}(Ci))\), where the mono-action transitions are directed to the final states representing the actions as illustrated in the previous examples. For impure collections see Subsection 2.6.3.

In practice, the algorithm is even easier and is constructed from Reverse by use of two functions 'stack-closure' and 'stack-successor', given in Appendix B.
2.6.3. Impure collections of stack automata

A collection is impure if it contains two articles

\[ A_i, \text{act} \, l, \quad A_j, \text{act} \, m, \]

where \( A_i \) and \( A_j \) are two initial articles of the look automaton and \( l \neq m \). If there is one impure state on one stack automaton, then the grammar is rejected (example in Appendix E).

2.7. Power of acceptance

The recognition power, depending of course on the cover used for the nonterminals, is a subset of Cohen-Culik's LRR with recognition of some LR(k) \((k \geq 1)\) grammars, as well as the recognition of some non-LR(k) grammars (cf. the proof in Section 2.10).

2.8. How to choose the regular cover for nonterminals?

The answer to the above question is not at all that evident and the choice has to be made according to one compulsory and two contradictory principles.

1. Above all, it seems absolutely necessary to conserve the valid-prefix property, i.e., only to take a shift decision for a valid prefix as to conserve the error-recovery routines of LR parsers. This is necessarily ensured if the context on the initial collection is the LALR(1) context. To obtain this, we only need to impose:

\[ \text{First1}(R\text{Cover}(I)) = \text{First1}(I). \]

The closure functions \( C_{lc} \) and \( C_{lp} \) definitely assure that the unicolumn transition matrix for the \( n \) actions on the initial context collection coincides with \( \bigcup C_1(\mu) \) for any \( \mu \) reaching the LR0 collection (cf. Section 3).

2. The number of possible states within a cover has to be sufficiently small to allow for an economical implementation. Computing the above has to be as simple as possible.

3. The cover should be tight enough to allow, if possible, for a discrimination of the actions through the first or the second part of the algorithm. Further, a mono-action output, as fast as possible, on the look automaton avoids any useless multiplication of the collections.

It can be easily understood that conditions (2) and (3) are not really satisfactorily fulfilled by one cover. We will offer two methods—the first satisfying correctly (1) and (2) and reasonably (3), the second being excellent for (3) and (1) and rather wearisome for (2).

2.8.1. Simple cover with types: SC(1)

This is the one we used for our implementation. The set of states is the set of pairs (terminal, type) where type describes the set \( \{ A, T, N, ] \} \); i.e., an implementation that works on three bits for the types and on at most one byte for the terminals.
(The flag—1 bit, the action code—4 bits, and the type—3 bits can be nicely implemented on 1 byte.)

The meaning of the types is as follows:

- **A** for all occurrences of the variable,
- **T** in front of a terminal,
- **N** in front of a nonterminal,
- **[** at the end of a production,
- **]** not at the end of a production, but either in front of a terminal or a nonterminal.

The type **merger** is as follows:

\[
(v, T \text{ and } v, N) \text{ or } (v, [ \text{ and } v, T) \text{ or } (v, [ \text{ and } v, N) \to V, [ \\
(v, x \text{ and } v, y) \to v, A]
\]

if \((x \in \{A, \}] \text{ and } y \in \{T, N, [\}) \text{ or } (x = A \text{ and } y = \}.

In the examples this information is found in the fields read and \(tv\).

By adapting the computing algorithm, “First” is given in five packs corresponding to the different types after merge. When entering on the cover of \(I\), we put the articles on the pairs \((v, \text{type})\) defined by “First”. Finally, the procedure for computing successors is simply

\[
\text{cover-succ(cover-closure(v, type, I))},
\]

where \(I\) designates the nonterminal concerned. cover-closure returns a list \(L\) of \((w, \text{type})\) where \(w\) is a nonterminal or a terminal, and cover-succ returns a list of \((w, \text{type})\) where \(w\) is a terminal. These functions are given in Appendix C.

Finally, a pair \((v, \text{type})\) is final if \(I, A\) is added in the closure. Please note that the case \(I \Rightarrow^* \varepsilon\) is directly taken into consideration by Clc. In spite of outward appearances, this cover is quite precise on a number of examples as shown in the examples of Appendix E and only sometimes they are too coarse. The implementation is simple and economical.

### 2.8.2. Adapting Boullier’s or Schimpf-Bermudez’s methods

Quite schematically, this method consists in using the LR0 automaton where the nonterminal transitions are suppressed and the reduce states linked to the \(\text{NEXT’s}\) with \(\varepsilon\)-transitions, with some subtleties that can be found in [11]. The adaptation requires

- choosing from among the LR0 collections one collection, denoted \(nI\), having one nucleus of the form \(W \Rightarrow \mu. \beta\) and a minimum of nuclei;
- taking, for the initial actions, just the ones from \(W \Rightarrow \mu. \beta\);
- \(dO(nI, I)\) being final;
- for the reductions, rejecting the predecessors which are not accessible from \(nI\) \((nI \text{ and these states are centers, so a closure on the centers suffices})\).

This cover has the advantage of being more precise than the previous ones, and the disadvantage of:
Yacc extension for LRR grammar parsing

- requiring the use of the LR0 automaton, which is not at all necessary for our module;
- demanding a closure on the centers in order to avoid switching errors on the reductions;
- and potentially using quite a number of states...

2.9. Minimization

This method provides two types of minimization options.

2.9.1. Option 1: look automata

When adding to the look automaton the same number of states as there are actions and stack automata, for the first example in Section 2.1 one obtains schematically the situation of Fig. 12.

First step: suppression of useless states on the look automaton. This means that all the states coaccessible of a single-stack state and none of the action states are to be suppressed. All the transitions reaching these states are reoriented on the stack automaton state. Figure 12 then results in Fig. 13. Because of the technique of deterministically merging \( n \) automata, this problem does not occur with the actions.

Second step: merging states. The recognized prefix languages on actions and stack states can be enlarged provided that no common prefixes are introduced and of course none taken out. Moreover, the context on the initial collection has to remain identical. These conditions are met if two states \( q_1 \) and \( q_2 \) are merged satisfying:

1. \( q_1 \) and \( q_2 \) are neither initial states nor actions or stack states;
2. for any \( a \in T \), \( d(q_1, a) = d(q_2, a) \) or \( d(q_1, a) = \emptyset \) or \( d(q_2, a) = \emptyset \).
It can be easily noted that the automaton remains deterministic and recognizes superlanguages after this operation. The difficulty of implementing a fast and economical procedure, a difficulty often encountered by automaticians when searching a minimal cover for an incompletely specified sequential network, leads us to consider that the merge is still the least negative of all solutions.

Third step: identification of the dead state with a noninitial and nonfinal state. Take the most visited state.

Fourth step: classical minimization. This technique provides excellent results as illustrated in example (3) of Section 2.1 where 125 collections of the look automaton were obtained, and merely two after minimization.

2.9.2. Option 1: stack automata

For the stack automata we proceed in an analogous way (steps (2), (3) and (4)). Moreover, the number of tables could still be reduced by identifying some tables inbetween the stack automata.

A disadvantage of this method could be the introduction of cycles where they did not exist initially, which leads us to define a second type of minimization (Option 2).

2.9.3. Option 2

This method consists in the first and fourth step above on all produced automata.

2.10. Summary of proof

We call one word of $\langle K C 0 C j \rangle$, where $C 0$ is the initial collection of the LR0, a viable prefix on a collection $C j$ of the LR0.
A specific action on a nucleus $n: Y \rightarrow \mu.\beta$ can be considered as a couple of dotted rules $(, Y \rightarrow \mu.\beta)$, where

(a) (direct actions) if $1: \beta \in T$ (direct shift), the first component is $Y \rightarrow \mu.\beta$, and if $\beta = \epsilon$ (reduce), the first component is $Y \rightarrow \mu$;

(b) (indirect actions) if $i: \beta = W \in N$, $W \ RO^* Z$ and $Z \rightarrow \epsilon \in P$ (reduce-$\epsilon$), then the first component is $Z \rightarrow \epsilon$; if $W \ RO^* Z$ and $Z \rightarrow a\mu 1 \in P$ (indirect shift), the first component is $Z \rightarrow a\mu 1$. From now on we will call an action on a nucleus $n$ a specific action on a nucleus $n$.

A cover of nonterminals is a substitution mapping ‘hr’ defined as

$$hr(a) = a \quad \text{if } a \in T,$$
$$hr(I) = R\text{cover}(I) \quad \text{if } I \in N,$$

where $R\text{cover}(I)$ is a regular set and a superset of $L(I)$.

In a rightmost derivation we will distinguish reaching an action on the left part, the frontier and the right part (see Figs. 14 and 15). If we modify the right-hand parts of the viable rightmost derivations reaching an action noted $acti$ of a nucleus noted $n$ in an LR0 collection $Cj$ by suppressing all the derivations trees, keeping the nonterminal roots, then we get a regular set on $N \cup T$, noted $R(\text{acti}, n, Cj)$. The right-context language of $(\text{acti}, n, Cj)$, noted $R\text{C}(\text{acti}, n, Cj)$, is obtained by the substitution ‘hl’ on $R(\text{acti}, n, Cj)$:

$$R\text{C}(\text{acti}, n, Cj) = hl(R(\text{acti}, n, Cj))$$

where $hl(a) = a$ and $hl(I) = L(I)$. Hence, the substitution on $R(\text{acti}, n, Cj)$ by ‘hr’, $hr(R(\text{acti}, n, Cj))$, is a regular cover of $R\text{C}(\text{acti}, n, Cj)$. The viability of the rightmost derivation is assured if the left part is a viable prefix of LR0 collection $Cj$. Furthermore, bottom-up, the derivation can be seen as a product of prefix-free ascent with left-spelling as illustrated in Figs. 14 and 15, where the right-hand part is supposed to be modified as above.

The pfa’s (short for prefix-free ascent) are of one of the three types shown in Fig. 15: for each of them we distinguish the left-hand part, the frontier, the modified right-hand part, the left-spelling, the right-hand word, the base and the objective. Types 2 and 3 are only used to get started.

Furthermore, one pfa belongs to a set of pfa’s which have the same base and the same objective, denoted PFA(base, objective). For the above modified derivation, we have

$$PFA_i: \quad PFA(Yi+1, Yi \rightarrow \mu_i.Wi\beta i) \quad \text{for } i \in [0, k-1];$$
$$PFA_k: \quad PFA(T, Yk \rightarrow \mu k.Wk\beta k) \quad \text{for direct action,}$$
$$PFA(Z \rightarrow a\delta, Yk \rightarrow \mu k.Wk\beta k) \quad \text{for indirect shift,}$$
$$PFA(Z \rightarrow \epsilon, Yk \rightarrow \mu k.Wk\beta k) \quad \text{for reduce}(Z \rightarrow \epsilon);$$
$$PFA_{k+1}: \quad PFA(W \rightarrow \mu k+1.\beta k+1, W \rightarrow \mu k+1.\beta k+1).$$
where the termination ACTION has one the two following forms

\begin{align*}
1 \text{-direct action} & \quad 2 \text{-indirect action} \\
\text{reduce or direct shift} & \quad \text{reduce-ε or indirect shift} \\
W_k & \quad W_k \\
. & \quad . \\
. & \quad . \\
. & \quad . \\
T. & \quad Z. \\
. & \quad . \\
. & \quad . \\
0 & \quad 0 \\
\text{left-spelling } = \pi T(\mu_i) & \quad \text{left-spelling } = \pi T(\mu_i) \\
_{k+1} & \quad _k \\
\end{align*}

Fig. 14.

For each product of pfai we get the same left-hand part if we change one pfai by another one belonging to the same PFAi. For each PFA we can define $R(PFA)$ as the regular set of all right-hand words of pfai elements, a set which can be derived from a left-linear grammar in an obvious way. The left-spelling is the common left-spelling. The product of PFA$(b_1, Y \rightarrow \mu, \beta)$PFA$(T, o_2)$ is consistent if

$$T = Y \quad \text{and} \quad o_2 \in \mathcal{D}_0^+ \left( (Y \rightarrow \mu, \beta), T(\mu) \right).$$

The above modified derivation of Fig. 14 is an element of $[\bigcup PFA_i$ for $i \in [k, 0]$ or $i \in [k+1, 0]$ depending on the type of the action.
We extend \( R(PFA) \) and left-spelling, in short 'ls', in a natural way to the products

\[
R(\prod PFA_i) = \prod R(PFA_i), \quad \text{ls}(\prod PFA_i) = \prod \text{ls}(PFA_i).
\]

It can be verified immediately that the regular set of words in the modified right-hand parts of the consistent product \( PFA1PFA2 \) coincides with \( R(PFA1)R(PFA2) \). We say that a consistent product of PFA's is viable if the left-spelling is the transpose of a viable prefix of the LR0's collection \( C_j \). On the viable products, which can be derived from a left-grammar by use of \( S_i \)'s and \( d_{pref} \), we obtain \( R(\text{acti}, n, C_j) \) as the set of

\[
R(\prod PFA_i) = \prod R(PFA_i),
\]

where \( \prod PFA_i \) is a viable product.

We can now summarize the proof of Algorithm 2 in four points.

(1) The underlying nondeterministic automaton for (action, \( n, C_j \))

\[
(A_i, dt-(C_{lc} \cup C_{lp}), A_0, A_f)
\]

exactly reads \( hr(R(\text{acti}, n, C_j)) \) on all viable products \( \prod PFA_i \) where the objects are
the quadruples of articles (we can skip the action code). $A_0$ is the initial article constructed as explained above. $Af$ is the final one: $(S' \rightarrow @, \text{dir}, 1, S' \rightarrow @)$, where we see the end of the algorithm on the look automaton as

$$dt[(S' \rightarrow \langle S \rangle), \text{dir}, 1, S' \rightarrow \langle \langle S \rangle \rangle)] = [S' \rightarrow \langle S \rangle],$$

$$(S' \rightarrow \langle S \rangle), \text{dir}, 1, S' \rightarrow \langle \langle S \rangle \rangle) \text{ Clp } (\langle S \rangle, 0, S' \rightarrow \langle \langle S \rangle \rangle),$$

$$(\langle \langle S \rangle \rangle, 0, S' \rightarrow \langle \langle S \rangle \rangle) \text{ Clp } (S' \rightarrow @, \text{dir}, 1, S' \rightarrow @).$$

It is easy to verify that we
- defined the initial PFA correctly;
- correctly simulated a PFAi ($dt$, Clc and the flag) by reading exactly $hr(R(PFAi))$ because correct progress is assured in the PFA by the use of $dt$ and Clc on the triples (context-component, {dir or q}, prefix-component) and because the flag allows getting out of the PFA at the right moment;
- passed from a PFAi to another one which can follow in a viable product by the use of dpref (Clp).

(2) The algorithm can also be seen as a sequential machine where:
- the first component is a context information on $T$,
- the second one is a prefix information on $N \cup T$,
- the transitions are $\varepsilon/\varepsilon$ for Clc, $a/\varepsilon$ for $dt$ and $\varepsilon/v$ for Clp,
- the states are the quadruples.

The first part is a subset construction on the first component with numbering of the articles, and the construction of crc and drp on the numbered articles. If $C_j$ is a collection of elements $A_j$ after the first subset construction, then, for $w$ belonging to $KC0C_j$, the transposes of all $\mu$ such that $w/\mu$ is read on the underlying nondeterministic sequential machine between $q0$ and $\text{Value}(A_j) = q_r$ are on $\text{drp-crc}(A_i, A_j, A_0)$, where $A_j$ is the initial state and $A_0$ the final one.

(3) The rest comes from the well-known technique of deterministically merging $n$ automata, where we use a unique code for all shifts.

(4) When the algorithm succeeds for all $LR_0(V)$ it is easy to prove that the grammar is LRR for the regular partition obtained by the refinement of all regular partitions obtained on $LR_0(V)$.

3. Some results on LR($k$) automata

It is indeed a tradition to present Knuth-LR($k$) as a subset construction on the items ($W \rightarrow \mu.\beta, u$) where $u \in T^*k$, but we chose to present it on the objects ($W \rightarrow \mu.\beta, L$) where $L \in \mathcal{P}(T^*k)$. In order to avoid disturbing the logicians, we will modify one line of the subset construction:

$$C_j = \text{Collect}(\text{Closure}(d(C_i, X))),$$

where the function 'Collect' is to unify $W \rightarrow \mu.\beta, Li$ in $W \rightarrow \mu.\beta, \cup Li$. Collections
that have been transformed in this way will be called unified collections. We have chosen for this point of view because this way is positive and enables us to rapidly obtain easy matrix equations on LR0, allowing for an evaluation of the copies induced on the LR(k). A summary of the above will be given in Proposition 3.1 below. Further, in Section 3.4, we will give an automaton for direct LRk(V) production analogous to the one we have presented for LR0(V).

3.1.

We introduce new sets of dotted rules: RDk designates the set of \((W \rightarrow \mu.\beta, L)\) where \(W \rightarrow \mu.\beta \in RD0\) and \(L \in \mathcal{P}(T^*k)\). The map ‘Core’ is defined on \(\bigcup RDk\), as usual:

\[
\text{Core}((W \rightarrow \mu.\beta), L) = (W \rightarrow \mu.\beta).
\]

When we restrict this mapping to RDk, we denote this as \(\text{Core}_k\). RNk, the set of nuclei of degree \(k\), is the set of pairs \((ni, L)\) where \(ni \in RN0\). When there is a possibility of confusion with the nuclei of degree 0, the nuclei of range \(k\) are denoted \(k-ni\). The set of closure rules of degree \(k\) is defined in the same way. On RDk we define the map ‘Ck’ by \(Ck((W \rightarrow \mu.\beta), L) = L\). Hence, we have \(k-ni = [\text{Core}(k-ni), Ck(k-ni)]\). If we extend Core and Ck on the collections of RDk, we consider a collection as a couple of one-column matrices, e.g.,

\[
S \rightarrow xV.c,\{\}\} \quad V \rightarrow V.a,\{a, c\} = \left[\begin{array}{c}
(S \rightarrow xV.c), \\
V \rightarrow V.a, \end{array}\right], \{\{\}\}, \{\{a, c\}\}.
\]

3.2.

We define, for each \(k\), a map PHIk on \(N \times N\) to \(\mathcal{P}(T^*k)\) as

\[
\text{PHI}_k(U, V) = \{w \in T^k| U \xrightarrow{rm}^* Vw \text{ and } w \in \text{First}_k(u)\}.
\]

PHIk can easily be designed in the following way: If, for each \(W \rightarrow V\beta \in P\) and for each \(w \in \text{First}_k(\beta)\), we establish a transition \(V \rightarrow^* W\), then we get an automaton on \(N\). PHIk(\(U, V\)) are all the \(k\)-prefixes we can obtain during a reading from \(V\) to \(U\).

We obviously have to change the transition system if we change the \(k\). For instance, for \(G0\) and for \(k = 1\) we get

\[
S \xrightarrow{e} P, \quad P \xrightarrow{e} F
\]

and

\[
\text{PHI}_1(S, S) = \{e, +\}, \quad \text{PHI}_1(F, F) = \{e\},
\]

\[
\text{PHI}_1(T, F) = \text{PHI}_1(T, T) = \{e, *\},
\]

\[
\text{PHI}_1(S, F) = \text{PHI}_1(S, T) = \{e, +, *\}.
\]
The use of \( \text{PHIk} \) is to collect the context on the internal lanes between the closure rules and the nuclei in a collection \( \text{LR}(k) \).

If we define the relation \( \text{PSI} \) by

\[
V \text{ PSI } U \text{ iff } U \rightarrow V \mu \text{ and } \mu \overset{*}{\Rightarrow} \varepsilon
\]

which is represented on every \( \text{PHIk} \) automaton by the \( e \)-transition system, then \( U \text{ PSI}^+ U \) is a sufficient condition for ambiguity (circularity). For \( \text{PHI1} \) we get

\[
a \in \text{PHI1}(U, V) \text{ iff } [V \text{ PSI}^* W \text{ and } W \overset{a}{\rightarrow} W_1 \text{ and } U \overset{\mu}{\rightarrow} W_1].
\]

3.3.

3.3.1. Results and applications

**Proposition 3.1.** (a) Knuth-LR\((k)\) is identified with

\[
d_k-\emptyset(\text{RNk}, N \cup T, ((S' \rightarrow \langle \cdot \rangle), \{\varepsilon\}), ((S' \rightarrow \langle \cdot \rangle), \{\varepsilon\}))
\]

where \( d_k \) is the union of two disjoint functions \( d_{sk} \) and \( d_{gk} \):

\[
d_{sk}((W \rightarrow \muX\beta), L, X) = ((W \rightarrow \muX\beta), L),
\]

\[
d_{gk}((W \rightarrow \muX\beta), L, X1) = \begin{cases} \emptyset & \text{if } X \in T, \\ \{(Z \rightarrow X1.\mu1, M) | X \overset{\mu}{\rightarrow} Z \text{ and } M = \text{PHIk}(X, Z) + k \text{ Firstk}(\beta) + k L\} & \text{otherwise}. \end{cases}
\]

where the collections are unified as explained above.

(b) For two unified and numbered collections of \( k \)-nuclei, where \( n^u \) designates the \( k \)-nuclei of \( N_i \), \( u \in \{1, p\} \), and \( n^v \) designates the \( k \)-nuclei of \( N_j \), \( v \in \{1, n\} \), we have

\[
N_j = d_k(N_i, X1) \text{ iff } \begin{cases} \text{d0(Core}(N_i), X1) = \text{Core}(N_j), \\ \text{Ck}(N_j) = M_j,i \text{ Ck}(N_i), \end{cases}
\]

where \( M_j,i \) is the \((n \times p)\)-matrix \( (a_{ij}^n) \) with \( a_{ij}^n \) defined in the following way:

\[
a_{ij}^n = \begin{cases} \{\varepsilon\} & \text{if } \text{Core}(n_{ij}^o) = \text{ds0(Core}(n_{ij}^o), X1), \\ \text{PHIk}(T, Z) + k \text{ Firstk}(\beta) & \text{if } \text{Core}(n_{ij}^o) \in \text{dg0[Core}(n_{ij}^o), X1] \\ \emptyset & \text{otherwise}. \end{cases}
\]

where the operators are union and concatenation limited to \( k \) characters.

(c) For the LR0 where the nuclei are numbered, if \( \text{d0} + (N0, \mu) = N' \) on the LR0, we define \( \text{Ck}(\mu) \) as follows: If the spelling of \( \mu \) on the LR0 is \( N0 = N_i0Ni1 \ldots Nin = N' \), then

\[
\text{Ck}(\mu) = \frac{1}{\prod_{j=n}^1 M_{j,j-1}} \text{ for } j \text{ ranging } [n, 1]
\]
where \( M_{j,j-1} \) is \((a_n^n)\) as defined above. Further, we then have

\[
\text{Core}_{k-1}(N') = \{(N', \text{Ck}(\mu)) \mid \mu / d0 + (N0, \mu) = N' \},
\]

where \( N0 \) is the initial collection of the LR0, and Core\(_{k-1}\) associates with \( N' \) the copies of the LR\(_k\) whose core is \( N' \).

(d) \( \text{LR}_k(V) \) is in bijection with the partition's elements of the equivalence \( \equiv_k \) defined on \( \text{KN}(V) \) by

\[
\mu_1 \equiv_k \mu_2 \iff \mu_1 \equiv_0 \mu_2 \text{ and } \text{Ck}(\mu_1) = \text{Ck}(\mu_2).
\]

Statements (b), (c), and (d) have the advantage of giving a direct equation of the \( \text{LR}_k(V) \) copies from \( \text{LR}_0(V) \) onwards and of defining the conditions for the \( \text{LR}_k(V) \) production.

We defer the proof of these propositions to the end of this section. We will now illustrate applications of them with the following examples.

(A) For Proposition 3.1(a): For the grammar

\[
\begin{align*}
S' &\to \langle \langle S \rangle \rangle, \\
S &\to xVc | xWd, \\
V &\to Va | a | X, \\
W &\to Wb | a | X, \\
X &\to t,
\end{align*}
\]

where PHI1 is given (as illustrated in Fig. 16) by

\[
\begin{align*}
\text{PHI}_1(V, X) &= \text{PHI}_1(V, V) = \{e, a \}, \\
\text{PHI}_1(S, S) &= \{e \}, \\
\text{PHI}_1(W, X) &= \text{PHI}_1(W, W) = \{e, b \},
\end{align*}
\]

Fig. 16.

the LR(1) automaton is obtained as shown in Fig. 17.

For instance, for the transition \( 4 \to 5 \), we get

\[
\text{dg}_k(\{(S \to x.Vc, \{\})\}, t) = (X \to t, \{a, c\})
\]

because \( \text{PHI}_1(V, X) = \{e, a \} \) and \( \text{First}_1(c) = \{c\} \) and, of course, \( \text{dg}_0(\{(S \to x.Vc)\}, t) = (X \to t) \). Further,

\[
\text{dg}_k(\{(S \to x.Wd, \{\})\}, t) = (X \to t, \{b, d\})
\]

because \( \text{PHI}_1(V, X) = \{e, b \} \) and \( \text{First}_1(d) = \{d\} \) and \( \text{dg}_0(\{(S \to x.Wd)\}, t) = (X \to t) \).

The two \( k \)-nuclei which have the same core are unified in \( X \to t, \{a, h, c, d\} \).
We get an immediate evaluation if we take, for all nuclei,

\[ C_k(n^*_j) = \bigcup (a^n_v + k \ C_k(n^*_j)) \]

where the \( a^n_v \) are defined as in Proposition 3.1(b) as we will explain now by the following example.

**Fig. 17.**

For Propositions 3.1(b), (c), and (d): For the grammar

\[ S' \rightarrow \langle S \rangle, \quad S \rightarrow BCB | AB, \]

\[ A \rightarrow aAbb | abb, \quad B \rightarrow aBb | ab, \]

\[ C \rightarrow Ca | a, \]

we extract from the LR0 the co-accessibles of state 2, as illustrated in Fig. 18.

For an evaluation of the LR(1) copies of state 2, we have the regular set of prefixes \( a^n b \) with \( n > 0 \).

If we construct the matrices, we get, for \( M_{2,1} \),

\[
\begin{array}{cccc}
11 & 12 & 13 & 14 \\
21 & \emptyset & \emptyset & \{\varepsilon\} & \emptyset \\
22 & \emptyset & \emptyset & \emptyset & \{\varepsilon\}
\end{array}
\]
For $M_{1,1}$ and $M_{1,0}$ we obtain respectively

<table>
<thead>
<tr>
<th></th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>01</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>{b}</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>∅</td>
<td>{b}</td>
<td>∅</td>
<td>∅</td>
<td>12</td>
</tr>
<tr>
<td>13</td>
<td>{b}</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>13</td>
</tr>
<tr>
<td>14</td>
<td>∅</td>
<td>{b}</td>
<td>∅</td>
<td>∅</td>
<td>14</td>
</tr>
</tbody>
</table>

If we realize that $(M_{1,1})^2 = M_{1,1}$, we find two possible answers for $C_1(\mu)$, namely $M_{2,1}M_{1,0}$ and $M_{2,1}M_{1,1}M_{1,0}$; that is

\[ C_1(\mu_1) = 21 \{b\} \quad C_1(\mu_2) = 22 \{a\} \]

with $\mu_1 \subseteq \{a^n b | n > 1\}$ and $\mu_2 = ab$. Finally, the two copies of state 2 on the LR1 are

- $B \to ab \{b\}$
- $A \to ab \{a\}$

Note that, for the cycle $M_c = (M_{i1}, i_2, M_{i2}, i_3 \ldots M_{ik}, i_1)$, we have to take the $p$ different situations $M_c, M_c^2, M_c^3, \ldots, M_c^p = M_c$.

### 3.3.2. Examination of a conflicting state

(Points (A), (B), and (C) below will show simplifications that can be added to the matrix examination of a conflict, and an application as an example.

(A) Matrix products are of course quite often saturated before $N_0$, which naturally calls for a definition of the pertinent suffixes by means of a frontier.

**Definition.** (i) A language of $\mathcal{P}(T^*k)$ is $k$-saturated if it contains words of length $k$ or ends with $\rangle$;
(ii) a matrix is \( k \)-saturated if its elements are \( \emptyset \) or \( k \)-saturated languages;
(iii) \( \text{COM} \) is the map that associates, with each \( k \)-saturated \((n \times p)\)-matrix \( M \), the one-column matrix where \( A'_i = \bigcup A_{i,j} \).

The frontier \( \text{Front} \) is defined as follows: for \( \mu / d_0 + (N_0, \mu) = N' \) and \( \mu : N_0 = N_1 N_2 \ldots N_n = N' \),

\[
\text{Front}_k(\mu) = \text{Front}_k(N_1 \ldots N_n) = 1
\]
such that \( C_k(N_1 \ldots N_n) \) is \( k \)-saturated and \( C_k(N_1 + 1 \ldots N_n) \) is not \( k \)-saturated. Then we have

\[
C_k(\mu) = \text{COM} \left( \bigcup_j M_{i,j-1} \right)
\]
for \( j \) ranging \([n, \text{Front}_k(\mu) + 1]\).

(B) A direct path between centers is a path of which the spelling is composed of centers at the edges, and of states that are not centers inbetween. For the grammars where there is just one direct path between one production center and the other, we can establish the matrix between the production centers because the intermediate states only shift the rule without changing the contexts.

(C) Furthermore, for the evaluation of a conflicting situation, we need another matrix, denoted \( \text{Dep}_k \) and constructed in the following way:
- a line is opened for each action, the shifts being gathered on the same line;
- the columns match the nuclei of the collection.

For each action its dependence on the nuclei concerned is expressed. More precisely, for column \( u \), corresponding to the nucleus \( nu \rightarrow \mu.X\beta \), we have the following three cases:

1. For the shift line:
   (a) direct connection \( X = a \in T \), one takes \( \text{First}_k(\alpha \beta) \);
   (b) indirect connection \( X \in N \) and \( X_1 \in T \rightarrow .X_1 \mu_1 \) with \( X \text{ RO}^* Z \), one takes
   \[
   \text{First}_k(X_1 \mu_1) + k \ \text{PHI}_k(X, Z) + k \ \text{First}_k(\beta)
   \]
   where \( a \) is the union of all words obtained and \( \emptyset \) otherwise.
2. For the reduce line \( Z \rightarrow \epsilon., \ X \in N \) with \( X \text{ RO}^* Z \), one takes
   \[
   \text{PHI}_k(X, Z) + k \ \text{First}_k(\beta)
   \]
   and \( \emptyset \) otherwise.
3. For a reduce line \( (W \rightarrow \mu) (X\beta = \epsilon) \), one takes \( \{\epsilon\} \) and \( \emptyset \) otherwise.

The new matrix product is then defined by \( C'_k(\mu) = \text{Dep}_k C_k(\mu) \).

All above remarks will be illustrated by the example of the grammar given in Appendix E.

For examination of state 6 for conflicting shift reduce, we extract from the LR0 the co-accessibles of state 6 as illustrated in Fig. 19. States 1, 3, and 5 are centers and there is just one direct path between them; the schematical situation for 6 is
given in Fig. 20. To examine state 6 with \( k = 1 \) we have as matrix \( \text{Dep1} \):

\[
\begin{array}{cccccc}
61 & 62 & 63 & 64 & 65 \\
\text{shift} & 0 & ( ) & ( ) & ( ) & ( ) \\
\text{reduce} & \{ e \} & 0 & 0 & 0 & 0 \\
\end{array}
\]

and as matrix products the tree of Fig. 21 where the matrices \( N \) are the condensed matrix products between centers. The matrix products are aborted as soon as they are saturated. Hence, we have two different \( C'k(\mu) \)'s

\[
\begin{align*}
\text{shift} & \quad , ( ) \\
\text{reduce} & \quad , ) )
\end{align*}
\]
The previous example implies that the state is neither LALR(1) nor LR(1), it is not LR(k) at all, but LRR as proven in one output list given in Appendix E.

3.4. Algorithm 3: k prefix automata

As for the LR0, we can construct an algorithm for the determination of LRk(V) by use of a transition function drk on the k-nuclei augmented with the new dotted rule S' → @, L as follows:

\[
drk[(W → μXβ, L), X] = \begin{cases} 
(W → μXβ, L) & \text{if } μ \neq ε, \\
\{(Z → μ1.Yβ1, M) | Y \text{ RO}^* W \text{ and } M = L + k \text{ PHIκ}(Y, W) + k \text{ Firstκ}(β1)\} & \text{otherwise}
\end{cases}
\]

for all \(ni\) belonging to RNk, except for the following case:

\[
drk[(S' → ⟨.S⟩, L), ⟨⟩] = (S' → @, L).
\]

If we simultaneously make the subset construction for the \(n\) nuclei of RN(V), we obtain the \(p\) states of LRk(V) as illustrated for G0 for RN(T) in Fig. 22.

The initial collection consists of triples

\[
S → S + T, 1, L1 = \{ε\}
\]
\[
S → T, 2, L2 = \{ε\}
\]
\[
S → T^*F, 3, L3 = \{ε\}
\]

Here we stop as soon as we have obtained the same set of indices for each rule as before and have \(k\)-saturated languages. (If a collection satisfies these two conditions,
it is a $k$-equidistributed collection.\) For a $k$-equidistributed collection such as

$$W \mapsto \alpha_1.X_1\beta_1, \{i_1 \ldots i_l\}, L_{i_1}^1 \ldots L_{i_l}^1$$

$$\vdots$$

$$W \mapsto \alpha_n.X_n\beta_n, \{i_1 \ldots i_l\}, L_{i_1}^n \ldots L_{i_l}^n,$$

we call the $m$ different $k$-tuples

$$Ik(Cf) = \{[(i_1, L_{i_1}^1), (i_2, L_{i_2}^1) \ldots (i_l, L_{i_l}^1)]\}$$

$k$-indicators. Further, by $k$-INDICATOR we denote the mapping which associates, with the initial collection, the $m$ different collections represented by the $k$-indicators. Here, for RN($T$), we have as $k$-indicators

$$[(1, \{\}, \{\}), (3, \{\{\}/+\})), [], [(1, \{+\}), (3, \{\}/+\}))],$$

$$[(2, \{\}, \{\}), (3, \{\{\}/+\})), [], [(2, \{+\}), (3, \{\}/+\}))].$$

We anticipate the proof of our third algorithm and conclude that, for LR1($T$), the two copies of

$$S \rightarrow T \quad \text{and} \quad T \rightarrow T.F$$
are
\[ S \rightarrow T, \{+/(+)\} \quad \text{and} \quad S \rightarrow T, \{+/(+)\} \]
\[ T \rightarrow T^*F, \{+/(^*)\} \quad \text{and} \quad T \rightarrow T^*F, \{+/(^*)\} \]

The two copies of
\[ S \rightarrow S + T \quad \text{and} \quad T \rightarrow T^*F \]
are
\[ S \rightarrow S + T, \{+/(+)\} \quad \text{and} \quad S \rightarrow S + T, \{+/(+)\} \]
\[ T \rightarrow T^*F, \{+/(^*)\} \quad \text{and} \quad T \rightarrow T^*F, \{+/(^*)\} \]

As before, we call the automaton obtained in the way as described above \textit{Autkpref-restricted} and the automaton produced by the same algorithm is called \textit{Autkpref}, where the notion of 'equidistributed' is restricted to

\[ S' \rightarrow @, \{il, \ldots, il\}, Li1, \ldots, Li_l. \]

As for the matrices we can adapt this to the scanning of a conflict in an obvious way. For instance, considering the following grammar:

\[ S \rightarrow aS | aAZ2c | aBZ2 | bA | bB, \]
\[ A \rightarrow c, \quad B \rightarrow cb, \]
\[ Z2 \rightarrow Z2b | b. \]

The analysis for \( k = 1 \) of the conflict \( A \rightarrow c \) (\( R1 \)) and \( B \rightarrow cb \) (\( S \)) results in Fig. 23. Here, \( \langle a^+ c \rangle \) and \( \langle a^*bc \rangle \) are two elements of the partition of the stack; one being 1-impure, the other one being LR(1).

| S1: A \rightarrow c., 1, R1 = \{e\} | S3: S \rightarrow aAZ2c, 1, R1 = \{b\} |
| S2: B \rightarrow c.b, 2, S = \{b\} | S4: S \rightarrow aBZ2, 2, S = \{b\} |
| S5: S \rightarrow b.A, 1, R1 = \{e\} | S6: S \rightarrow b.B, 2, S = \{b\} |
| S7: S \rightarrow a.S, 12, R1 = \{b\}, S = \{b\} | S10: S \rightarrow a.S, 12, R1 = \{e\}, S = \{b\} |
| S8: S \rightarrow \langle\langle .S \rangle\rangle, 12, R1 = \{b\}, S = \{b\} | S11: S \rightarrow \langle\langle .S \rangle\rangle, 12, R1 = \{e\}, S = \{b\} |
| S12: S' \rightarrow @, R1 = \{\rangle\rangle\}, S = \{b\} |

Fig. 23.
Proof of Algorithm 3.

\[
LR_k(\mathcal{V}) = k\text{-INDICATOR}(\text{Autkpref-restricted}(\mathcal{R}(\mathcal{V}))) = k\text{-INDICATOR}(\text{Autkpref}(\mathcal{R}(\mathcal{V})))
\]

Proof. For a final collection \( C' \) of \( \text{Autkpref}(\mathcal{R}(\mathcal{V})) \), where the \( k \)-indicator is \( \{i_1, \ldots, i_l\} \), \( \{L_i_1, L_i_2, \ldots, L_i_l\} \), we have that \( \{i_1, \ldots, i_l\} \) is 0-indicator of \( \text{Aut0pref}(\mathcal{R}(\mathcal{V})) \) on collection \( C' \), e.g., an element of \( \text{LR0}(\mathcal{V}) \). If we define \( C_k(n_i, \Gamma) \) for \( \Gamma \in K_{ni} \) as follows: if \( W \rightarrow \mu.\beta \) designates \( n_i \), then

\[
C_k(n_i, \Gamma) = \{ w \in T^*k \left| S' \xrightarrow{\delta W x} w \text{ and } w \in \text{First}(x) \text{ and } \Gamma = \delta \mu \}\}
\]

then, by induction, we get that \( Liu = C_k(n_iu, \Gamma) \) for all \( \Gamma \) belonging to \( T(KCOCf) \), and \( u \in [1, l] \). \( \square \)

Hence, Algorithm 3 produces the partition of \( \bigcup K_{ni} \) on the equivalence \( =_k \) on the \( C' \), and therefore determines \( LR_k(\mathcal{V}) \) according to Proposition 3.1. As a conclusion to the proof, we make the same comment now for \( \text{Autkpref-restricted} \) as we made for \( \text{Aut0pref-restricted} \) in Section 1.

3.5. Proof of Proposition 3.1

(a): From the definitions we can identify the initial states of \( \text{LR0} \) and of \( \text{LR}(k) \). The two states are denoted \( N_0 \). As for \( \text{LR0} \) we define a new closure \( Cl_k \) on the nuclei onto the closure rules in order to suppress the superfluous Knuth closures. The definition is as follows:

\[
Cl_k(W \rightarrow \mu X\beta, L) = \begin{cases} 
\emptyset & \text{if } X \in T, \\
\{(Z \rightarrow \mu 1, M) | X \text{ RO } Z, M = \text{PHI}_k(X, Z) + k \text{ First}_k(\beta) + k L\} & \text{otherwise}.
\end{cases}
\]

From the Knuth-closure process we get, for a chain that starts on a nucleus,

\[
W \rightarrow \mu X_0\beta_0, a_0 \quad \text{KCl } X_0 \rightarrow X_1\beta_1, a_1 \quad a_1 \in \text{First}_k(\beta_0a_0),
\]

\[
X_0 \rightarrow X_1\beta_1, a_1 \quad \text{KCl } X_1 \rightarrow X_2\beta_2, a_2 \quad a_2 \in \text{First}_k(\beta_1a_1),
\]

\[
\vdots
\]

\[
X_{n-2} \rightarrow X_{n-1}\beta_{n-1}, an \rightarrow X_{n}\beta, an \quad an \in \text{First}_k(\beta n - 1an - 1),
\]

where \( an \) belongs to

\[
\text{First}_k(\beta n - 1\beta n - 2 \ldots \beta 1) + k \text{ First}_k(\beta_0) + k \{a_0\} \quad \text{and} \quad X_0 \text{ RO } X_n - 1
\]

but \( \text{First}_k(\beta n - 1 \ldots \beta 1) \) is included in \( \text{PHI}_k(X_0, X_n - i) \). Hence, \( an \) belongs to \( \text{PHI}_k(X_0, X_n - 1) + k \text{ First}_k(\beta_0) + k \{a_0\} \).
From the definition of \( \text{PHIk}(X, Z) \) it immediately appears that, for all \( Z \) such that \( X \to^* Z \) all the \( Z \to^\alpha b \) where \( b \) belongs to \( \text{PHIk}(X, Z) + k \text{ Firstk}(\beta) + k L \), are in the \( \text{KCl}^* \) of \( (W \to^\mu X\beta, L) \). Hence, Knuth-closure(\( Ci \)) = Clk(\( Ci \)) for all collections of \( k \)-nuclei.

If we suppress the \( \varepsilon \)-transitions of

\[
\text{N DFA}[\text{RDk}, V, dsk \cup \text{Clk}, ((S' \to \langle S \rangle), \{\varepsilon\} (\langle S' \to \langle S \rangle), \{\varepsilon\})])
\]

we get \( \text{N DFA}[\text{RNk}, V, dsk \cup dgk \ldots] \) and, finally, the DFA \( dk-\emptyset[\ldots] \).

(b): Statement (a) of Proposition 3.1 leads us to observe that \( dgk(Ci) \cap dsk(Ci) = \emptyset \) for all collections of \( k \)-nuclei (same remark as for \( ds0 \) and \( dg0 \)). From the definitions of \( ds0 \), \( dg0 \), \( dsk \), \( dgk \) we immediately obtain

\[
dgk(ni, X) = nj \Rightarrow dg0(\text{Core}(ni), X) = \text{Core}(nj)
\]

\[
dsk(ni, X) = nj \Rightarrow ds0(\text{Core}(ni), X) = \text{Core}(nj)
\]

if \( dk(\text{Ni}, X) = Nj \). We examine the unified collection \( \text{Nj} \):

- if the nucleus \( nv \) is shifted from \( nu \in \text{Ni} \) to \( dsk(nu, X) = nv \), we have \( Ck(nv) = Ck(nu) \);
- if the nucleus \( nv \) is generated from \( nu1, nu2, \ldots, nup \) in \( \text{Ni} \), we have \( Ck(nv) = \bigcup Ck(dgk(nul, X)) \) for \( l \in \{1, \ldots, p\} \); hence, \( \bigcup (a^u_0 + k Ck(nul)) \). As \( \emptyset + k L = \emptyset \), we have, for all \( nv \),

\[
Ck(nv) = \bigcup (a^u_0 + k Ck(nu)).
\]

Hence, \( Ck(\text{Nj}) = \prod M_{j,i} Ck(\cdot;i)(1) \).

Now, for a state \( Nl\text{r}0 \) which is different from the initial state of \( \text{LR0} \), let us call \( Nc1, Nc2, \ldots, Ncn \) the states of some core on \( \text{LR}(k) \). The definitions will show a well-known fact, which is that, for any \( \mu / d0+(N0, \mu) = Nl\text{r}0 \), there is one and only one \( Nci \) such that \( dk+(N0, \mu) = Nci \) on the \( \text{LR}(k) \). All the \( \mu \) such that \( dk+(N0, \mu) = Ncj \) verify that \( d0+(N0, \mu) = Nl\text{r}0 \) on the \( \text{LR0} \).

By recurrence on the length of \( \mu \) we obtain from (1) that if \( dk+(N0, \mu) = Ncj \) (where the spelling of \( \mu \) is \( N0Ni1Ni2 \ldots N\text{in} = Ncj \) on the \( \text{LR}(k) \) and \( N0N'i1N'i2 \ldots N'in = Nl\text{r}0 \) on the \( \text{LR0} \), then\n
\[
Ck(Ncj) = \prod M_{j,i-1} \quad \text{for } j \text{ ranging } [n, 1],
\]

where \( M_{j,i-1} \) is evaluated on the \( N'i'j \) on the \( \text{LR0} \). Consequently, \( \{Ck(Ncj)\} \) coincides with \( \{Ck(\mu)\} \) evaluated on the \( \text{LR0} \). Therefore, we indeed obtain \( \text{Corek-1}(Nl\text{r}0) = \{(Nl\text{r}0, Ck(\mu))\} \).

Finally, from the last demonstration, one immediately concludes that the set \( \text{LRk}(V) \) is deduced from the one of \( \text{LR0}(V) \) by joining the condition \( Ck(\mu1) = Ck(\mu2) \) to the equivalence \( =_0 \). The initial state is considered to be the representative of \( \langle \cdot \rangle \), with which the context \( \varepsilon \) is associated. \( \square \)
Yacc extension for LRR grammar parsing

Appendix A. Clc ∪ Clp(Ai)

switch(type-article){
    case prefixe:
        switch(type-prefixe){
            case stable:
                Sl = dpref(Sj) /* just one in this case */
                create(("",left-label(Sj),",Sl,"),i)
                break;
            default:
                for each Sl belonging to dpref(Sj)
                    /* If Z is the left non terminal of Sj */
                    prefix-free-ascent(Z,Sl,i)
                endfor
                }endswitch type-prefixe
        break;
    case contexte:
        switch(type-contexte){
            case completely-dead:
                create(("",left-label(Sj),"O,Sj,`),i)
                break;
                /* we use the second component to keep the stack transition */
            case half-dead:
                /* If 2 is the left non terminal of the context-component*/
                prefix-free-ascent(Z,Sj,i)
                break;
            case alive:
                switch(type-reading){
                    case direct:
                        /* If the context-component is Z → µ,YB */
                        switch(X){
                            case terminal:
                                break;
                        case non terminal:
                            If X → µ
                                create((Z → µ,X,B,dir,\ldots,),i)
                            endif
                            create((Z → µ,X,B,qo,\ldots,),i)
                            /* where qo is the initial state of the regular cover of X */
                            /* in fact it's better to wait the transitions to put the article
                            in cover mode */
                            break;
                        }endswitch
                        default:
                            /* the article is in indirect reading mode
                                (Z → µ,I,B,q,\ldots,) ou I ∈ N et q one state of Rcover(I) */
                            If q is a final state of Rcover(I)
                            create((Z → µ,I,B,dir,\ldots,),i)
                        endif
                        }endswitch
                    }endswitch type-contexte
                break;
            default:
                /* the article is in indirect reading mode
                 (Z → µ,I,B,q,\ldots,) ou I ∈ N et q one state of Rcover(I) */
            }endswitch type-contexte
        break;
    }endswitch type-article

prefix-free-ascent(Z,Sl,i):

    /* We suppose Sl to have the form of W → µ,TB */
    If Z = T
    create((W → µ,T,B,dir,1,W → µ,T,B,),i)
    endif
    /* the flag indicates that we begin to read the context of the prefix-component */
    For each Y → Z8I belonging to G / TB*/ Y
    create((Y → Z8I,dir,0,W → µ,T,B,),i)
    endfor
    /* end of prefix-free-ascent */
The function ‘create’ checks whether the article is in the stack or not, and updates internal Reverse.

Appendix B. Stack-closure({Bi})

Initial List : unmarked Bi
while there exist an unmarked article Bj :An,action
mark it
For each Bl belonging to Reverse(Bj)
/* Bl : Al,action */
switch(type-Al){
case prefixe:
add-if-necessary (Al,action) marked
break;
case contexte:
add-if-necessary (Al,action) unmarked
}endswitch
endfor
endwhile
stack-successor({Bj),v)
For each Bj
/* Bj: An,action */
switch(type-Aj){
case contexte:
break;
case prefixe:
If the second component of An not equal to v
break;
endif
For each Al belonging to Reverse(An)
add-if-necessary (Al,action)
endfor
}endswitch
endfor

Appendix C. Cover-closure(v, type, I)

initial list : (v,type) unmarked
while there exist an unmarked (v,type)
mark it
for each grammatical occurrence of v compatible with (v,type) on the accessibles.
rules from I
/* e.g W \rightarrow \mu v.XXI \beta */
If XXI \beta = \varepsilon add-if-necessary (W,A) 
If X \in N et X \Rightarrow \varepsilon add-if-necessary(XI,type')
endfor
cndwhile

In ‘add-if-necessary’ a merging of types is performed if necessary.
cover-Succ({w,type,I})
For each element of {w,type,I} :(v,type)
For each grammatical occurrence of v compatible with (v,type) on the accessibles
rules from I
W \rightarrow \mu v.XY \beta
switch(X){
case terminal:
    add-if-necessary (X,type') (type' is N if \( Y \in N \), T if \( Y \in T \)
    and \( ] \) if \( YB = g. \))
    break;

case non terminal:
    add-if-necessary all (w,type') in First(X).
}
endfor
endfor

In ‘add-if-necessary’ a merging of types is performed if necessary.

Appendix D. Grammar from example (3) in Section 2.1

**GRAMMAR RULES**

0) \( \text{S}' \rightarrow << \text{P} >> \)
1) \( \text{P} ightarrow \text{S} \)
2) \( \text{P} ightarrow \text{P} \); \( \text{S} \)
3) \( \text{S} ightarrow \text{AS} \)
4) \( \text{S} ightarrow \text{J} \)
5) \( \text{S} ightarrow \text{ID} \); \( \text{S} \)
6) \( \text{AS} ightarrow \text{ID} = \text{E} \)
7) \( \text{AS} ightarrow \text{ID} \text{ eq} \text{SE} \)
8) \( \text{J} ightarrow \text{if R then goto ID} \)
9) \( \text{R} ightarrow \text{E} = \text{E} \)
10) \( \text{R} ightarrow \text{SE eq SE} \)
11) \( \text{E} ightarrow \text{E} + \text{T} \)
12) \( \text{E} ightarrow \text{E} - \text{T} \)
13) \( \text{E} ightarrow \text{T} \)
14) \( \text{T} ightarrow \text{T} * \text{F} \)
15) \( \text{T} ightarrow \text{F} \)
16) \( \text{F} ightarrow ( \text{E} ) \)
17) \( \text{F} ightarrow \text{ID} \)
18) \( \text{F} ightarrow \text{CS} \)
19) \( \text{SE} ightarrow \text{SE} + \text{ST} \)
20) \( \text{SE} ightarrow \text{ST} \)
21) \( \text{ST} ightarrow \text{ST} * \text{SF} \)
22) \( \text{ST} ightarrow \text{SF} \)
23) \( \text{SF} ightarrow \text{SF} - \text{SG} \)
24) \( \text{SF} ightarrow \text{SG} \)
25) \( \text{SG} ightarrow ( \text{SE} ) \)
26) \( \text{SG} ightarrow \text{ID} \)
27) \( \text{SG} ightarrow \text{CS} \)
28) \( \text{ID} ightarrow \text{ID} \text{ L} \)
29) \( \text{ID} ightarrow \text{L} \)
30) \( \text{CS} ightarrow \text{CS} \text{ DI} \)
31) \( \text{CS} ightarrow \text{DI} \)
32) \( \text{L} ightarrow \text{a} \)
33) \( \text{L} ightarrow \text{b} \)
34) \( \text{DI} ightarrow \text{0} \)
35) \( \text{DI} ightarrow \text{1} \)

Appendix E

E.1. Excerpt from ADA grammar where “#” stands for “/", one non-LR(\(k\)) conflict and its resolution
E.2. A trivial grammar, neither XLR nor R(h)LR0 nor LAR(h), with arbitrary reading of stack

**GRAMMAR RULES**

\[ S' \rightarrow \text{<< } S \text{ >>} \]

1) \( S \rightarrow dS \)

2) \( S \rightarrow aTa \)

3) \( S \rightarrow bTb \)

4) \( S \rightarrow aUb \)

5) \( S \rightarrow bUa \)

6) \( T \rightarrow aTa \)

7) \( U \rightarrow aUa \)

8) \( T \rightarrow c \)

9) \( U \rightarrow c \)

**ANALYSIS FOR VARIABLE c**

\( \text{LR0 variable c no: 000} \)

\( T \rightarrow c \)

\( U \rightarrow c \)

\( \text{reduce } 8 \)

\( \text{shift } \)

**SLR look reduce 009: a b**

**SLR look reduce 009: a b**

**ACtIONS CODE**

**shift 0**

**reduce 5**

**NOT LALR(1)**

- Articles: 120
- Collections: 12
- Space used: 600 bytes

**MINIMIZATION LOOK AUTOMATON type 2**

**Ax stands for action x**

**Refer to the codes**

\( \text{look } 009: a b \)

\( \text{look } 009: a b \)

\( \text{look } 009: a b \)

\( \text{look } 009: a b \)
**E.3. A simple LR(1) grammar not accessible by previous methods is limited to a look examination and a very compact solution**

**GRAMMAR RULES**

0) $S' \rightarrow << S >>$
1) $S \rightarrow x W a$
2) $S \rightarrow y W b$
3) $S \rightarrow z W r$
4) $S \rightarrow x V$
5) $S \rightarrow y V$
6) $S \rightarrow z V$
7) $S \rightarrow u U X d$
8) $S \rightarrow u U Y E$
9) $W \rightarrow U X C$
10) $V \rightarrow U Y d$
11) $U \rightarrow U t$
12) $U \rightarrow s$
13) $X \rightarrow t U X P$
14) $X \rightarrow t$
15) $Y \rightarrow t U Y u$
16) $Y \rightarrow t$
17) $E \rightarrow a$
18) $E \rightarrow b$
19) $E \rightarrow c$
20) $E \rightarrow v$
21) $C \rightarrow c$
22) $C \rightarrow w$
23) $C \rightarrow$
24) $P \rightarrow$

**ANALYSIS FOR VARIABLE t**

LR0 COLLECTIONS:

LR0 variable t no: 000

U → U t _
X → t _ U X P
X → t _
Y \rightarrow t_\_ U Y u
Y \rightarrow t_\_
shift a
reduce 11 14 16

INADEQUATE
slr look reduce 011: t
slr look reduce 014: a b r d c w
slr look reduce 016: a b u d c v

ACTIONS CODE
red 11 0
shift 1
red 14 2
red 16 3
NOT LALR(1)

Articles: 83
Collections: 5
Space used: 415 bytes

STACK AUTOMATON NO S00 (LOOK COLLECTION A001)
Articles: 4
Collections: 1
Space used: 12 bytes

STACK AUTOMATON NO S01 (LOOK COLLECTION A002)
Articles: 4
Collections: 1
Space used: 12 bytes

STACK AUTOMATON NO S02 (LOOK COLLECTION A003)
Articles: 6
Collections: 1
Space used: 18 bytes

MINIMIZATION LOOK && STACK AUTOMATA type 2
** Ax stands for action x && Sx for Stack Automaton x **
** Refer to the codes **

SWVUXYECPC < x a y b z r u d <

Look
LOC01
LOC02
Stack
SOC01
Stack
S1C01
Stack
S2C01

E.4. One failing example on a non-LRR and nonambiguous grammar

GRAMMAR RULES
0) S' \rightarrow \langle S \rangle
1) S \rightarrow B C B
2) S \rightarrow A B
Yacc extension for LRR grammar parsing

3) A → a A b b
4) A → a b b
5) B → a B b
6) B → a b
7) C → C a
8) C → a

LR0 variable b no: 004
A → a b _ b
B → a b _
shift b
reduce 6

INADEQUATE

s1r look reduce 006: a b >>
S1 VALUE dpref
30 A → a b _ b
S1 B → a b _
S2 A → a _ b b
S3 B → a _ b
S4 S' → << _ S >>
S5 A → a _ A b b
S6 B → a _ B b
S7 S' → @

ACTIONS CODE
shift 0
red 6 1

NOT LALR(1)

LOOK COLLECTION NO: A000

TRANSITIONS:
b --> 1

LOOK COLLECTION NO: A001

TRANSITIONS:
a --> 2
b --> 3

LOOK COLLECTION NO: A002

TRANSITIONS:
a --> 4

LOOK COLLECTION NO: A003
<table>
<thead>
<tr>
<th>TRANSITIONS:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a --&gt; 4</td>
<td></td>
</tr>
<tr>
<td>b --&gt; 6</td>
<td></td>
</tr>
</tbody>
</table>

**LOOK COLLECTION NO: A004**

<table>
<thead>
<tr>
<th>no context or ~ var</th>
<th>read tv f pref ac</th>
<th>internal Reverse</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 S --&gt; B C B</td>
<td>a</td>
<td>0 S4 1</td>
</tr>
<tr>
<td>01 S --&gt; B C B</td>
<td>dir</td>
<td>0 S4 1</td>
</tr>
<tr>
<td>02 S --&gt; B C B</td>
<td>a</td>
<td>0 S4 1</td>
</tr>
<tr>
<td>03 S --&gt; A B B</td>
<td>a</td>
<td>0 S4 0</td>
</tr>
</tbody>
</table>

**TRANSITIONS:**

| a --> 4     |  |
| b --> 6     |  |

**LOOK COLLECTION NO: A005**

<table>
<thead>
<tr>
<th>no context or ~ var</th>
<th>read tv f pref ac</th>
<th>internal Reverse</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 S --&gt; B C B</td>
<td>dir</td>
<td>0 S4 1</td>
</tr>
<tr>
<td>01 S --&gt; A B B</td>
<td>dir</td>
<td>0 S4 0</td>
</tr>
<tr>
<td>02 A --&gt; a A b b</td>
<td>dir</td>
<td>1 S5 0</td>
</tr>
<tr>
<td>03 A --&gt; a A b b</td>
<td>dir</td>
<td>1 S5 0</td>
</tr>
<tr>
<td>04 B --&gt; a B b b</td>
<td>dir</td>
<td>1 S6 1</td>
</tr>
<tr>
<td>05 B --&gt; a B b b</td>
<td>dir</td>
<td>1 S6 1</td>
</tr>
<tr>
<td>06 ~ a</td>
<td>0 S5 0</td>
<td></td>
</tr>
<tr>
<td>07 ~ a</td>
<td>0 S6 1</td>
<td></td>
</tr>
</tbody>
</table>

**TRANSITIONS:**

| a --> 2     |  |
| b --> 3     |  |

**LOOK COLLECTION NO: A006**

<table>
<thead>
<tr>
<th>no context or ~ var</th>
<th>read tv f pref ac</th>
<th>internal Reverse</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 S' --&gt; &lt;&lt; S &gt;&gt;</td>
<td>dir</td>
<td>1 S4 0</td>
</tr>
<tr>
<td>01 S' --&gt; &lt;&lt; S &gt;&gt;</td>
<td>dir</td>
<td>1 S4 1</td>
</tr>
<tr>
<td>02 S --&gt; B C B</td>
<td>b</td>
<td>0 S4 1</td>
</tr>
<tr>
<td>03 S --&gt; B C B</td>
<td>dir</td>
<td>0 S4 1</td>
</tr>
<tr>
<td>04 S --&gt; A B B</td>
<td>b</td>
<td>0 S4 0</td>
</tr>
<tr>
<td>05 S --&gt; A B B</td>
<td>dir</td>
<td>0 S4 0</td>
</tr>
</tbody>
</table>

**TRANSITIONS:**

| IMPURE b --> 6 |

**MULTICTIONS**

- Articles: 41
- Collections: 7
- Space used: 205 bytes

**IMPURE LOOK --- STACK NEXT PAGE**

**STACK AUTOMATON NO S00 (LOOK COLLECTION A006)**

**STACK COLLECTION NO: B000**

<table>
<thead>
<tr>
<th>no A...... action</th>
</tr>
</thead>
<tbody>
<tr>
<td>000 A001,01</td>
</tr>
<tr>
<td>001 A001,06</td>
</tr>
<tr>
<td>002 A002,02</td>
</tr>
<tr>
<td>003 A004,03</td>
</tr>
<tr>
<td>004 A005,01</td>
</tr>
<tr>
<td>005 A005,06</td>
</tr>
<tr>
<td>006 A006,00</td>
</tr>
<tr>
<td>007 A006,04</td>
</tr>
<tr>
<td>008 A006,05</td>
</tr>
<tr>
<td>009 A001,00</td>
</tr>
<tr>
<td>010 A001,07</td>
</tr>
<tr>
<td>011 A002,00</td>
</tr>
<tr>
<td>012 A002,01</td>
</tr>
<tr>
<td>013 A004,00</td>
</tr>
<tr>
<td>014 A004,01</td>
</tr>
<tr>
<td>015 A004,02</td>
</tr>
<tr>
<td>016 A005,00</td>
</tr>
<tr>
<td>017 A005,07</td>
</tr>
<tr>
<td>018 A006,01</td>
</tr>
<tr>
<td>019 A006,02</td>
</tr>
</tbody>
</table>
Yacc extension for LRR grammar parsing

020 A006,03 1
TRANSITIONS:
a --> 1
STACK COLLECTION NO: B001
no A action
000 A001,02 0
001 A001,06 0
002 A001,08 0
003 A003,01 0
004 A005,02 0
005 A005,03 0
006 A005,06 0
007 A000,02 1
008 A000,04 1
009 A001,05 1
010 A003,02 1
011 A003,04 1
012 A005,05 1
shift b
TRANSITIONS:
a --> 2
STACK COLLECTION NO: B002
no A action
000 A001,06 0
001 A001,08 0
002 A003,01 n
003 A005,02 0
004 A005,03 0
005 A005,06 0
006 A000,05 1
007 A000,04 1
008 A001,04 1
009 A001,07 1
010 A003,03 1
011 A005,04 1
012 A005,07 1
TRANSITIONS:
a --> 1
b --> 3
STACK COLLECTION NO: B003
no A action
000 A000,01 0
001 A001,03 0
002 A000,03 1
MULTI ACTIONS
Articles : 50
Collections : 4
Space used : 150 bytes

IMPURE - SORRY - STOP

Appendix F. Summary of a simple interconnection with Yacc

- Find on y.output.c the non-LALR(1) states;
- a value less than YYFLAG in yydef[] enables to trace the non-LALR(1) states, and to take a default reduction decision;
- independent production of decision automata by Algorithm 2;
- insertion in y.tab.c of these automata as well as a new function lrr.
  lrr is used when yydef[yystate] <= YYFLAG and the call looks like
  
yyn = lrr(yystate, yyyps, & yys[1]).

yyn is the number of reduction chosen or 0.
  A new 'goto yyreduce' allows to transfer control of reduction processing if yyn > 0.
  - A new yylexl() allows for bufferization of tokens;
  - stack reading is easily implemented because of yycheck[].
  
N.B.: This scheme supposes that one gives up disambiguity rules to recover shift actions in yypact and yyact.

Acknowledgment

I would like to express my gratitude to Mr. Nivat for his assistance with my University Thesis which gave birth to this present paper, and to Mr. Aho and Mr. Pager for the kind attention they paid to my work.

References

Yacc extension for LRR grammar parsing

[27] M. Mickunas, R. Lancaster and V. Schneider, Transforming LR(k) grammars to LR(1), SLR(1), and (1, 1) BRC grammars, J. ACM 23(3) (1976) 511–533.
[34] B. Setié, Une extension de Yacc, pour analyse des grammaires LRR, qui autorise un examen non borné du contexte droit et de la pile, Ph.D. Thesis, University of Paris VII, April 1986.