Theoretical Computer Science 82 (1991) 165-176
Elsevier

Note

A general context-free parsing algorithm running in linear time on every LR(k) grammar without using lookahead

Joop M.I.M. Leo*

University of Nijmegen, Department of Computer Science, Toernooiveld, 6525 ED Nijmegen, Netherlands

Communicated by M. Harrison
Received February 1987
Revised October 1989

Abstract

Leo, J., A general context-free parsing algorithm running in linear time on every LR(k) grammar without using lookahead, Theoretical Computer Science 82 (1991) 165-176.

A new general context-free parsing algorithm is presented which runs in linear time and space on every LR(k) grammar without using any lookahead and without making use of the LR property. Most of the existing implementations of tabular parsing algorithms, including those using lookahead, can easily be adapted to this new algorithm without a noteworthy loss of efficiency. For some natural right recursive grammars both the time and space complexity will be improved from $\Omega(n^2)$ to $O(n)$. This makes this algorithm not only of theoretical but probably of practical interest as well.

1. Introduction

General context-free parsing and recognition methods are used in applications for which the known linear-time methods are too restricted. The main areas where such general methods are extensively used are systems for the processing of natural languages, speech recognition, and in compiler generating systems. The known

* Present affiliation: Oracle Corporation/Oracle Nederland, Rijnzathe 6, 3454 PV De Meern, Netherlands.

general parsing methods make use of backtracking or they are tabular in nature. None of them is completely satisfactory. Those making use of backtracking have the advantage of running in linear space, but their time complexity can be exponential in the length of the input for some grammars. (For a rather large class of grammars, however, even the simple topdown backtrack parsers run in linear or low polynomial time [9].)

On the other hand, the tabular algorithm of Earley [3, 4] and the improved tabular algorithm of Graham, Harrison and Ruzzo [5, 6] run for arbitrary context-free grammars in time $O(n^3)$ and space $O(n^2)$, where n is the length of the input. For unambiguous grammars they run in time and space $O(n^2)$ and for non-right recursive LR(k) grammars in linear time and space even if no lookahead is used. For some right recursive LR(k) grammars however, (such as $S \rightarrow aS \mid \lambda$.), they need a lookahead of k to obtain a linear time and space performance. If not enough lookahead is used, the time complexity, and, what is even worse, the space complexity for such grammars is $\Omega(n^2)$. Also for non-LR(k) grammars right recursion is sometimes the reason for a bad performance of these parsers. As certain syntactic structures in both programming languages and natural languages are more naturally described by right recursive constructs, this is a serious weakness of these methods.

In his thesis, Earley [3, p. 60] conjectured that by a slight change in the algorithm a linear-time performance for every LR(k) grammar can be obtained without the use of lookahead. In this paper we will show that his conjecture is correct. Since the LR property itself plays no role for the steps taken by our algorithm, the LR(k)-ness does not have to be determined. Furthermore, the modification does not lead to a noteworthy loss of efficiency for any grammar, and the resulting algorithm can be combined easily with lookahead.

2. Informal description

We use the following notational conventions. Let $G = (V, \Sigma, P, S)$ denote an arbitrary context-free grammar, where V is the vocabulary, Σ the set of terminal symbols, P a finite set of productions, and S the start symbol. $V - \Sigma$ is the set of non-terminal symbols, denoted by N. Let $w = a_1 \ldots a_n$ with $a_i \in \Sigma$ $(1 \leqslant i \leqslant n)$ denote the input string to be parsed or recognized. The substring $a_{i+1} \ldots a_j$ is denoted as w_{ij}. Instead of w_{0j} we also write w_j.

A *recognizer* is a procedure to decide for every w in Σ^* whether or not $w \in L(G)$. A *parser* is a recognizer which yields for every $w \in L(G)$ the derivation(s) of w in one form or another.

Before we present our algorithm, we first give a brief description of the recognition algorithm of Cocke, Kasami, and Younger (the CKY-algorithm) [7, 8, 12], the algorithm of Earley [3, 4], the algorithm of Graham, Harrison and Ruzzo [5, 6], and Tomita's algorithm [11]. Except Tomita's algorithm, they are all dynamic programming methods.

The CKY-algorithm only works for grammars in Chomsky normal form. It builds an upper triangular matrix (t_{ij}) in which a non-terminal A is put into t_{ij} $(0 \leq i < j \leq n)$ iff $A \overset{*}{\Rightarrow} w_{ij}$.

Earley's method works for every context-free grammar, so no grammar transformations are required. In its simplest form, where no lookahead is used, it successively builds lists $I_0, \ldots, I_n$ containing items of the form $[A \to \alpha.\beta, i]$ where $A \to \alpha\beta$ is a production and . is a symbol not in V. When list I_j is completed, it contains item $[A \to \alpha.\beta, i]$ iff $S \overset{*}{\Rightarrow} w_i A\gamma$ for some $\gamma \in V^*$ and $\alpha \overset{*}{\Rightarrow} w_{ij}$. (The formulation in terms of lists can be regarded as a suitable representation for a recognition matrix.)

The algorithms of Graham et al. [5] (except for their theoretically interesting, but impractical $O(n^3/\log n)$ algorithm) differ from Earley's in three respects: the order in which the items are computed, the choice of the data structure, and the precomputation of empty and chain derivations. These changes often mean a considerable improvement of the constant factors, but the asymptotic time and space complexities are unchanged.

Tomita's algorithm works for every cycle-free context-free grammar. It is based on LR parsing. Multiple entries in the LR parsing table are handled by pseudo-parallel parsing in different directions. By using certain sharing techniques, the parsing time and space are polynomial, although not always $O(n^3)$.[1]

For recognition purposes only, the space complexity of Earley's algorithm could be improved in some cases by removing all complete items (i.e. items with the dot at the end of a production) in I_j after this list has been built, but in general this would make parsing harder. Nevertheless, as we will show, some of the complete items contributing to a parse can easily be reconstructed afterwards in a deterministic way. Moreover, some complete items need not be generated at all while building the lists $I_0, \ldots, I_n$. For certain right recursive grammars this will speed up the recognition time by a factor n. How this can be accomplished is sketched in the following.

For reasons of presentation we shall, for the moment, assume that the grammar does not contain non-terminals which can only produce the empty string λ as terminal word.

Initialization: Let I_0 be the set of all items of the form $[A \to \alpha.\beta, 0]$ with $A \to \alpha\beta$ a production in P, such that $S \overset{*}{\Rightarrow} A\eta$ for some $\eta \in V^*$, and $\alpha \overset{*}{\Rightarrow} \lambda$.

Now assume that the sets $I_0, \ldots, I_{j-1}$ have already been constructed, and set I_j is still empty.

Scanner: First, for each item $[A \to \alpha.a_j\delta\gamma, i] \in I_{j-1}$ such that $\delta \overset{*}{\Rightarrow} \lambda$, we add $[A \to \alpha a_j\delta.\gamma, i]$ to I_j.

[1] It can be proved that the time and space complexity of Tomita's algorithm is $O(n^l)$ where l is the maximum length of the productions of the grammar, i.e. $l = \max_{A \to \alpha \in P} \lg(A\alpha)$.

Completer: Next, for each complete item of the form $[A \to \gamma., i]$ in or newly added to I_j, we add, if it exists, the topmost complete item on the deterministic reduction path above $[A \to \gamma., i]$ to I_j (see Definition 2.1). If it does not exist, then for each item $[B \to \alpha.A\delta\eta, k] \in I_i$ such that $\delta \stackrel{*}{\Rightarrow} \lambda$, we add $[B \to \alpha A\delta.\eta, k]$ to I_j.

Predictor: Finally, for each item $[A \to \alpha.B\beta, i] \in I_j$, we add to I_j all items of the form $[C \to \gamma.\xi, j]$ with $C \to \gamma\xi \in P$, $\gamma \stackrel{*}{\Rightarrow} \lambda$, and $B \stackrel{*}{\Rightarrow} C\eta$ for some $\eta \in V^*$.

Definition 2.1. An item is said to be *on the deterministic reduction path above* $[A \to \gamma., i]$ if it is $[B \to \alpha A., k]$ with $[B \to \alpha.A, k]$ being the only item in I_i with the dot in front of A, or if it is on the deterministic reduction path above $[B \to \alpha A., k]$. An item on such a path is called the *topmost* one if there is no item on the deterministic reduction path above it.

On a (non-empty) deterministic reduction path there always exists a topmost item if $S \stackrel{+}{\Rightarrow} S$ is impossible. The easiest way to avoid problems in this respect is to augment the grammar with a new start symbol S', i.e. adding the rule $S' \to S$ to the grammar, with S' not in V, and letting S' be the new start symbol.

To determine the topmost item on a deterministic reduction path we do not always have to construct the complete reduction path. Suppose $[C \to \delta., m]$ is the topmost item on the deterministic reduction path above $[A \to \gamma., i]$, and $[B \to \beta., k]$ is some other item on this path. Then we add to set I_k a so-called *transitive item* $[C \to \delta., B, m]$. Subsequently, if an item of the form $[B \to \beta'., k]$ is added to some set $I_{j'}$, we can directly add $[C \to \delta., m]$ to $I_{j'}$.

When the sets $I_0, \ldots, I_n$ have been constructed, then the input w is in $L(G)$ iff $[S' \to S., 0]$ is in I_n (or, if the grammar is not augmented, some item of the form $[S \to \alpha., 0]$ is in I_n).

If we apply the construction just given to certain LR(k) grammars with non-terminals which can only produce λ, (like $S \to aSE \mid \lambda$. $E \to \lambda$), the time and space complexities remain $\Omega(n^2)$. Therefore, for the general case, we will treat, in the construction of the deterministic reduction paths, items of the form $[A \to \alpha.B\delta, i]$ for which δ can only produce λ as terminal word just like items of the form $[A \to \alpha.B, i]$. (If δ is empty, then we also say that δ produces empty.)

We illustrate the algorithm described here with an example.

Example 2.2. Consider the grammar

$S \to aS \mid C.$

$C \to aCb \mid \lambda.$ and input $a^n b^m$ $(n \geq m)$.

Earley's algorithm builds:

$I_0 = \{[S \to .aS, 0], [S \to .C, 0], [S \to C., 0], [C \to .aCb, 0], [C \to ., 0]\}$ #initialization
for $1 \leq i \leq n$;

$I_i = \{[S \to a.S, i-1], [S \to aS., i-1], [C \to a.Cb, i-1], [C \to aC.b, i-1],$ #scanner
$[S \to aS., i-2], [S \to aS., i-3], \ldots, [S \to aS., 0],$ #completer
$[S \to .aS, i], [S \to .C, i], [S \to C., i], [C \to .aCb, i], [C \to ., i]\}$ #predictor
for $n < i \leq n+m$;

$I_i = \{[C \to aCb., 2n-i],$ #scanner
$[C \to aC.b, 2n-i-1], [S \to C., 2n-i],$ #completer
$[S \to aS., 2n-i-1], [S \to aS., 2n-i-2], \ldots, [S \to aS., 0]\}.$ #completer

Our algorithm builds:

$I_0 = \{[S \to .aS, 0], [S \to .C, 0], [S \to C., 0], [C \to .aCb, 0], [C \to ., 0]\}$ #initialization
for $1 \leq i \leq n$;

$I_i = \{[S \to a.S, i-1], [S \to aS., i-1], [C \to a.Cb, i-1], [C \to aC.b, i-1],$ #scanner
$[S \to aS., 0],$ #completer
$[S \to .aS, i], [S \to .C, i], [S \to C., i], [C \to .aCb, i], [C \to ., i],$ #predictor
$[S \to aS., S, 0]\}$ #trans. item
for $n < i \leq n+m$;

$I_i = \{[C \to aCb., 2n-i],$ #scanner
$[C \to aC.b, 2n-i-1], [S \to C., 2n-i], [S \to aS., 0]\}.$ #completer

We see that instead of adding $[S \to aS., i-2], [S \to aS., i-3], \ldots, [S \to aS., 1]$ to the sets I_i $(1 \leq i \leq n)$ as in Earley's method we only add the transitive item $[S \to aS., S, 0]$.

Note that for certain LR languages, like for instance the language $\{a^n b^m \mid n \geq m\}$ considered in this example, every LR(k) grammar is right recursive, and that Earley's method runs for each of these LR(k) grammars in time and space $\Omega(n^2)$ if insufficient lookahead is used.

The algorithm loosely described in this section runs in linear time and space for every LR(k) grammar, without using any lookahead. In the next section we give a more precise description of our algorithm.

3. The algorithm

Definition 3.1. Let $G = (V, \Sigma, P, S)$ be a CFG, and let . be a symbol not in V. If $A \to \alpha\beta \in P$, then $A \to \alpha.\beta$ is called a *dotted rule*, $[A \to \alpha.\beta, i]$ with i an integer representing an index in the input string is called an *item*, and $[A \to \alpha\beta., B, i]$ with $B \in N$ is called a *transitive item*.

Definition 3.2. Let $[A \to \alpha.\beta, i]$ be an item. If $\beta = \lambda$, then the item is called *complete*. If $\beta \neq \lambda$, then the item is called *incomplete*. If β can only produce the empty string

λ as a terminal word, then we call the item *quasi-complete.* Otherwise we call it *strongly incomplete.* (Note that the definition is such that complete items are also quasi-complete.)

The next two definitions are related to those found in [5] and [6].

Definition 3.3. Let $G = (V, \Sigma, P, S)$ be a CFG, let Q be a set of items, and let $X \in V$. Define $Q \times X = \{[A \to \alpha X \beta . \gamma, k] \mid [A \to \alpha . X \beta \gamma, k] \in Q, \beta \stackrel{*}{\Rightarrow} \lambda\}$.

Definition 3.4. Let $G = (V, \Sigma, P, S)$ be a CFG, let $R \subseteq V - \Sigma$, and j an index. Define $\text{PREDICT}_j(R) = \{[C \to \gamma . \xi, j] \mid C \to \gamma\xi \in P,\ \gamma \stackrel{*}{\Rightarrow} \lambda$, and $B \stackrel{*}{\Rightarrow} C\eta$ for some $B \in R$ and some $\eta \in V^*\}$.

As stated in the previous section, the existence of a topmost item on a deterministic reduction path is not always guaranteed in case $S \stackrel{+}{\Rightarrow} S$. Therefore, in the next algorithm the grammar has been augmented with S' as the new start symbol.

Algorithm 1
begin
  $I_0 :=$ PREDICT$_0(\{S'\})$;
  for $j := 1$ **to** n
  do
    $I_j := I_{j-1} \times a_j$;
    for each item of the form $[A \to \gamma., i]$ in or newly added to I_j
    do
      T_UPDATE$(I_0, \ldots, I_i, A)$;
      if I_i contains a transitive item of the form $[B \to \beta., A, k]$
      then $I_j := I_j \cup \{[B \to \beta., k]\}$
      else $I_j := I_j \cup I_i \times A$
      fi
    od;
    $I_j := I_j \cup \text{PREDICT}_j(\{A \mid [B \to \alpha . A \beta, k] \in I_j\})$
  od;
  if $[S' \to S., 0]$ is in I_n **then** accept **else** reject **fi**
end.

T_UPDATE$(I_0, \ldots, I_i, A)$:
  global *t_rule*, *t_pos*;
  if I_i contains a transitive item of the form $[B \to \beta., A, k]$
  then *t_rule* $:= B \to \beta.$; *t_pos* $:= k$
  elif I_i contains exactly one item of the form $[B \to \alpha . A \beta, k]$ **and**
    $[B \to \alpha A . \beta, k]$ is quasi-complete
  then *t_rule* $:= B \to \alpha A \beta.$; *t_pos* $:= k$;
    T_UPDATE$(I_0, \ldots, I_k, B)$;
    $I_i := I_i \cup \{[$*t_rule*, A, *t_pos*$]\}$
  fi.

Remark 3.5. (1) The two global variables *t_rule* and *t_pos* used in the procedure *T*_UPDATE do not have to be initialized.

(2) If in Algorithm 1 the **do**-part of the inner **for**-loop is replaced by $I_j := I_j \cup I_i \times A$, then the sets I_j are exactly the same as in Earley's algorithm.

(3) For some grammars Algorithm 1 requires more space than Earley's algorithm. For example, for $S \to ASb \mid \lambda$. $A \to aA \mid c$. and input $(ac)^n b^n$ the sets I_j built by Earley and by Algorithm 1 are equal with respect to the "normal" items, but in our method the sets I_{2i+1} $(0 \leq i < n)$ additionally contain the transitive item $[A \to aA., A, 2i]$. However, if we do not add a transitive item $[A \to \alpha B., B, k]$ to I_i if I_i also contains $[A \to \alpha.B, k]$, then the total number of items and transitive items added by our algorithm never exceeds the number of items added by Earley's. The required modification of Algorithm 1 is simple. Note that adding a transitive item may speed up the recognition time, but not adding it will not affect the correctness of the algorithm.

(4) In the description of the algorithm we have abstracted from the representation of the sets I_j. By elementary list handling techniques, the operations and tests in the algorithm can be performed without extensive searches. A rather good choice for the data structure for I_j is to use for each X in V a list containing all items in I_j of the form $[A \to \alpha.X\beta, i]$, and if present $[A \to \alpha., X, k]$. The completed items in I_j could be kept in another list. In [5] a few other efficient implementations are discussed, which are applicable to our algorithm as well.

3.1. Parsing and reducing

Up to now we have only considered context-free recognizers, but normally we are more interested in parsers. After the sets $I_0, \ldots, I_n$ have been built, and the sentence recognized, we could either construct the parses directly, or we could transform the sets $I_0, \ldots, I_n$ to the "normal" reduced ones of Earley, i.e. transform them such that afterwards $[A \to \alpha.\beta, i] \in I_j$ iff $S \Rightarrow^* w_i A \gamma$ for some $\gamma \in V^*$ and $\alpha \Rightarrow^* w_{ij}$ and $\beta\gamma \Rightarrow^* w_{jn}$ (the so-called *useful items*). This transformation has the advantage that other routines taking the "normal" sets as input need no modification, but the disadvantage that for certain ambiguous grammars, like

$$S \to AA'. \qquad A \to aA \mid a. \qquad A' \to aA' \mid b.$$

the space requirements grow from linear to quadratic.

The transformation can be done by marking the useful items in a similar way as described in [5]. The difference with the reduction algorithm in [5] is that we still have to add the useful items that have been omitted. For arbitrary grammars the transformation time is at worst comparable to that required for reducing the "normal" lists of Earley; for unambiguous grammars the time costs are roughly comparable to the time required by Algorithm 1.

4. Theoretical results

Most of the results given in this section are not really difficult to prove, but the proofs are sometimes long and tedious. Therefore, a sketch of the proof is given only in those cases where the results are not intuitively plausible.

Theorem 4.1. *Algorithm* 1 *accepts* w *iff* $w \in L(G)$.

The following lemma is useful for the proof of this theorem.

Lemma 4.2. (a) *Algorithm* 1 *adds a strongly incomplete item* $[A \to \alpha.\beta, i]$ *to* I_j *iff* $S \overset{*}{\Rightarrow} w_i A\gamma$ *for some* $\gamma \in V^*$ *and* $\alpha \overset{*}{\Rightarrow} w_{ij}$.

(b) *If a transitive item* $[B \to \beta., C, m]$ *is added to* I_k *for some* $k < j$, *then for each* $\gamma \in V^*$; $A \in N$; $i \leq m$, *if*

(i) $S \overset{*}{\Rightarrow} w_i A\gamma$,

(ii) $A \overset{*}{\Rightarrow} w_{ik} C$,

then $A \overset{*}{\Rightarrow} w_{im} B \Rightarrow w_{im}\beta \overset{*}{\Rightarrow} w_{ik} C$.

Proof. Parts (a) and (b) can be proved simultaneously by induction on j. □

Thus the sets built by Algorithm 1 and by Earley's do not differ with respect to the strongly incomplete items.

Theorem 4.3. *The order of magnitude of the time and space complexity of Algorithm* 1 *is for no grammar worse than that of Earley's algorithm.*

The use of lookahead in our algorithm might lower the order for certain (ambiguous) grammars. For example, for the grammar $S \to Ab.$ $A \to aA \mid aaA \mid \lambda.$, Earley's algorithm and ours without lookahead both take time and space $\Omega(n^2)$, whereas with a lookahead of 1 they both take time and space $O(n)$.

In [2] the class of LR-regular grammars was introduced. It forms a direct generalization of the LR(k) grammars. Instead of using only a bounded lookahead to determine the handle in a right sentential form, a regular lookahead is used. As we will show, Algorithm 1 runs in linear time even for this extension of the class of the LR(k) grammars.

Definition 4.4. A partition $\pi = \{B_1, \ldots, B_m\}$ of Σ^* is called *regular* if all the sets B_i are regular. ($\{B_1, \ldots, B_m\}$ is called a partition of Σ^* if the union of the B_i's is equal to Σ^*, and the B_i's are mutually disjoint.)

Definition 4.5. Let $G = (V, \Sigma, P, S)$ be a reduced context-free grammar such that $S \overset{+}{\Rightarrow} S$ is not possible in G. G is LR(π) with $\pi = \{B_1, \ldots, B_m\}$ some partition of Σ^* if, for each $w, w', x \in \Sigma^*$; $\alpha, \alpha', \beta, \beta' \in V^*$; $A, A' \in N$, if

(i) $S \underset{R}{\overset{*}{\Rightarrow}} \alpha A w \underset{R}{\Rightarrow} \alpha\beta w$,

(ii) $S \overset{*}{\Rightarrow}_R \alpha' A' x \Rightarrow_R \alpha' \beta' x = \alpha\beta w'$,
(iii) $w \equiv w' (\text{mod } \pi)$,
then $(A \to \beta, \lg(\alpha\beta)) = (A' \to \beta', \lg(\alpha'\beta'))$.

A context-free grammar is called *LR-regular* if it is LR(π) for some regular partition π of Σ^*.

Every LR(k) grammar is an LR(π) grammar with the regular partition

$$\pi = \{\{vw \mid w \in \Sigma^*\} \mid v \in \Sigma^k\} \cup \{\{v\} \mid v \in \Sigma^*, \text{and } \lg(v) < k\}.$$

Now we state our main theorem.

Theorem 4.6. *Algorithm* 1 *runs in linear time and linear space for every LR-regular grammar.*

Proof (*sketch*). First we show, in the next three lemmas, that for LR-regular grammars the number of items and the number of transitive items in each set I_j is bounded by some constant. It follows that the space complexity is linear in the length of the input. Using this result, it then follows from Lemma 4.10 that the time complexity is linear as well.

Lemma 4.7. *For every LR-regular grammar, there exists a constant c such that the number of strongly incomplete items in each set I_j is at most c.*

Proof. Without loss of generality we assume that the partition $\pi = \{B_1, \ldots, B_m\}$ of Σ^* is a *left congruence* (i.e. for every x, y, z in Σ^*, $x \equiv y (\text{mod } \pi)$ implies $zx \equiv zy (\text{mod } \pi)$). Now let $A \to \beta\gamma$ be a production in P for which γ can produce some $v \in \Sigma^+$. Then, for every $\alpha, \alpha' \in V^*$; $u, w, w' \in \Sigma^*$, if
(i) $S \overset{*}{\Rightarrow}_R \alpha A w \Rightarrow_R \alpha\beta\gamma w \overset{*}{\Rightarrow}_R \alpha\beta v w \overset{*}{\Rightarrow} uvw$,
(ii) $S \overset{*}{\Rightarrow}_R \alpha' A w' \Rightarrow_R \alpha'\beta\gamma w' \overset{*}{\Rightarrow}_R \alpha'\beta v w' \overset{*}{\Rightarrow} uvw'$,
(iii) $w \equiv w' (\text{mod } \pi)$,
then $\alpha = \alpha'$. (Proof omitted.)

Since LR-regular grammars are unambiguous, the factorization of u in $u_1 u_2$ such that $\alpha \overset{*}{\Rightarrow} u_1$ and $\beta \overset{*}{\Rightarrow} u_2$ is unique. Therefore, there are for each j at most m different i's such that $[A \to \beta . \gamma, i]$ is in I_j. □

Lemma 4.8. *For every unambiguous grammar, there exist constants c, d such that the number of quasi-complete items in each set I_j is at most c times the number of strongly incomplete items in I_j plus d.*

Proof. Since the number of items added by PREDICT$_j$ are at most c_1, and the number of quasi-complete items in I_j is at most c_2 times the number of complete

items in I_j, with c_1 and c_2 some constants determined by the grammar, it is sufficient to prove that there exist constants c, d such that the number of complete items $[A \to \alpha., i]$ in each set I_j with $i < j$ is at most c times the number of strongly incomplete items in I_j plus d.

The key to the proof lies in the following observation.

Observation. Consider the items found in one execution of the **do**-part of the inner **for**-loop of the algorithm. Assume that exactly one complete item is found, and no strongly incomplete item. Assume further that this complete item is new and unequal to $[S' \to S., 0]$. Then, in a following execution of the loop applied to this newly added item, there will be found at least one strongly incomplete item, or at least two complete items.

Now let $[A \to \alpha., i]$ be some complete item in I_j with $i < j$, and let $[A \to \alpha., i]^*$ denote the set containing $[A \to \alpha., i]$ and all items $[B \to \beta.\gamma, k]$ in I_j for which $\beta \stackrel{*}{\Rightarrow} w_{ki}A$. Let $n_{\text{internals}}$ be the number of complete items different from $[S' \to S., 0]$ in $[A \to \alpha., i]^*$ and n_{leaves} be the number of strongly incomplete items or $[S' \to S., 0]$ in $[A \to \alpha., i]^*$. Using the observation and the fact that for unambiguous grammars only new items are found, it can be proved by (structural) induction that $n_{\text{internals}} \leq 4n_{\text{leaves}} - 2$.

Because, for unambiguous grammars, two arbitrary sets $[A \to \alpha., i]^*$ and $[A' \to \alpha'., i']^*$ of items in I_j are either disjoint or one is contained in the other, a similar relation can be obtained for the union of such sets. The rest of the proof of Lemma 4.8 is now straightforward. □

Lemma 4.9. *For every grammar, there exists a constant c such that the number of transitive items in each set I_j is at most c.*

Proof. Each set I_j contains for each non-terminal A at most one transitive item of the form $[B \to \beta., A, i]$. □

From the next lemma it follows that the time performance for LR-regular grammars is linear as well.

Lemma 4.10. *For every unambiguous grammar, the time and space complexities of Algorithm 1 have the same order of magnitude.*

Proof. Each elementary test of the algorithm and each addition of an item or transitive item can, if efficiently implemented, be performed in time bounded by some constant (cf. [5, pp. 439-441]). It is not difficult to see that the number of steps taken by the algorithm is proportional to the number of items (including duplicates) and transitive items found. □

Remarks 4.11. (1) Lemma 4.8 is not true for every ambiguous grammar. Not even for certain grammars whose degree of direct ambiguity is 1. For example, the grammar $S \to aS \mid aaS \mid \lambda$. has degree of direct ambiguity equal to 1, but for input a^n the number of quasi-complete items in I_j ($j \geq 1$) is $2j$ (or $2j+1$ if we augment the grammar with a new start symbol), whereas the number of strongly incomplete items in I_j is only 5.

(2) The class of grammars for which Algorithm 1 runs in linear time properly includes the class of LR-regular grammars. There are (even infinitely) ambiguous grammars (e.g. grammar (1) below), as well as unambiguous grammars which are not LR-regular (e.g. grammar (2) below), for which our algorithm and Earley's algorithm run in linear time.

(1) $S \to Sa \mid Saa \mid \lambda$.

(2) $S \to X \mid Y$. $X \to aXb \mid ab$. $Y \to aYbb \mid \lambda$.

Note that grammar (2) is not LR(π) for any finite partition π of Σ^*, and that no equivalent LR-regular grammar exists for it. (By the way, every unambiguous grammar is an LR(π) grammar with π the infinite partition $\{\{w\} \mid w \in \Sigma^*\}$, and every LR($\pi$) grammar with π some finite or infinite partition of Σ^* is also unambiguous.) Although we have not (yet) found a proof, we conjecture that Algorithm 1 runs in linear time for every LR(π) grammar, with π some finite, but not necessarily regular, partition of Σ^*.

It is undecidable for an arbitrary grammar whether Algorithm 1 runs on it in linear time (or linear space). Algorithm 1 runs in linear time and linear space on

$$S \to R_1 S \mid R_2 S \mid \lambda.$$

$$R_1 \to x_i R_1 d c^i \mid x_i d c^i. \quad \text{for } 1 \leq i \leq m,\ x_i \in \{a, b\}^+,$$

$$R_2 \to y_i R_2 d c^i \mid y_i d c^i. \quad \text{for } 1 \leq i \leq m,\ y_i \in \{a, b\}^+,$$

iff $L(R_1) \cap L(R_2) = \emptyset$. Whether this intersection is empty is recursively unsolvable, since the Post Correspondence Problem is recursively unsolvable.

5. Conclusion

Our algorithm is of interest in that it makes lookahead unnecessary for obtaining a linear time performance for parsing every LR(k) grammar. This does, however, not mean that lookahead has completely lost its value, as it may still lower the constant factors considerably, and for at least some ambiguous grammars even the order.

We have not yet tested our algorithm extensively, but so far its performance is often significantly better (and never worse) than the performance of Earley's algorithm. By using a simple kind of lookahead and by early elimination of those

items which are not consistent with the next input symbol, we expect the algorithm to become really practical for various applications. A first version has been successfully incorporated in a translator generator, which is used for the processing of natural as well as programming languages [10].

Acknowledgment

I wish to thank C.H.A. Koster, H. Meijer, and A. Nijholt for valuable suggestions and fruitful conversations about the subject, and one of the referees for detailed comments.

References

[1] A.V. Aho and J.D. Ullman, *The Theory of Parsing, Translation, and Compiling, Vol. 1: Parsing* (Prentice Hall, Englewood Cliffs, NJ, 1972).

[2] K. Culik II and R. Cohen, LR-regular grammars—an extension of LR(k) grammars, *J. Comput. System Sci.* **7** (1973) 66-96.

[3] J. Earley, An efficient context-free parsing algorithm, Ph.D. Thesis, Carnegie Mellon University, Pittsburgh, PA, 1968.

[4] J. Earley, An efficient context-free parsing algorithm, *Comm. ACM* **13** (2) (1970) 94-102.

[5] S.L. Graham, M.A. Harrison and W.L. Ruzzo, An improved context-free recognizer, *ACM Trans. Programming Languages Systems* **2** (1980) 415-462.

[6] M.A. Harrison, *Introduction to Formal Language Theory* (Addison-Wesley, Reading, MA, 1978).

[7] D. Hays, Automatic language-data processing, in: H. Borko, ed., *Computer Applications in the Behavioral Sciences* (Prentice Hall, Engewood Cliffs, NJ, 1962) 394-423.

[8] T. Kasami, An efficient recognition and syntax analysis algorithm for context free languages, University of Illinois, 1966.

[9] J.M.I.M. Leo, On the complexity of topdown backtrack parsers, in: *Proc. NGI-SION Symp.*, Utrecht, The Netherlands (1986) 343-355.

[10] H. Meijer, Programmar: a translator generator, Ph.D. Thesis, University of Nijmegen, The Netherlands, 1986.

[11] M. Tomita, *Efficient Parsing for Natural Languages* (Kluwer, Hingham, MA, 1986).

[12] D.H. Younger, Recognition and parsing of context free languages in time n^3, *Inform. and Control* **10** (1967) 189-208.