



## Gap Parsing with LL(1) Grammars

EBERHARD BERTSCH  
Ruhr University, Bochum  
Faculty of Mathematics  
Universitätsstraße 150, D-44780 Bochum  
Germany  
E-mail: [bertsch@lpi.rub.de](mailto:bertsch@lpi.rub.de)

MARK-JAN NEDERHOF  
University of Groningen  
Faculty of Arts  
P.O. Box 716, 9700 AS Groningen  
The Netherlands  
E-mail: [markjan@let.rug.nl](mailto:markjan@let.rug.nl)

### Abstract

We study the time complexity of the parsing problem for input strings that contain gaps. For context-free languages the time requirement is cubic. For restricted classes of languages given by LL(1), bidirectional LL(1), and a specific kind of XML grammars, the time requirement is bounded linearly or quadratically depending on the number of separate gaps.

## 1 Introduction

We consider input strings that are *incomplete*, i.e. that contain unknown substrings at specified locations. The unknown substrings will be referred to as *gaps*. An incomplete string specifies an infinite set of *completed* strings, each of which results by choosing a substring for each gap. The main problem we address is how to decide whether any of these infinitely many completed strings are in the language generated by a context-free grammar. In particular we study the time complexities for particular bounds on the number of gaps and for particular classes of context-free grammars. We will concentrate on LL(1) grammars and bi-LL(1) grammars, i.e. grammars that are LL(1) in both directions. We show:

- For an arbitrary number of gaps and for an arbitrary CFG, the time complexity is cubic.
- For a single gap at the beginning of the input and an LL(1) grammar, the time complexity is linear.

- For a single gap at any position and an LL(1) grammar, the time complexity is quadratic.
- For a single gap at any position and a bi-LL(1) grammar, the time complexity is linear.
- For two gaps and a bi-LL(1) grammar, the time complexity is quadratic.
- The last two results carry over to XML grammars.

The problem of a single gap at the beginning of the input has also been called *suffix parsing*. Partial solutions for the class of operator precedence grammars were found by, amongst others, [14]. For LL(1) grammars, a linear-time algorithm was found by [5]. For LR(1) grammars, a linear-time *recognition* algorithm was found by [3], and a linear-time *parsing* algorithm by [13]. The practical relevance of efficient suffix parsing is immediately evident from previous work on non-correcting error recovery [14, 7, 18, 10]. The other problems mentioned above may play a role in the incremental development of programs using a non-structural editor, where the user may indicate a number of textual gaps to be filled in later.

The algorithms we propose for bi-LL(1) grammars rely on a new parsing technique, consisting in “bridging” the gap between two substrings that have been previously processed by two variants of Earley’s algorithm, the first operating left-to-right and the second right-to-left. This parsing technique leads to novel results of computational complexity that cannot be achieved by standard unidirectional parsing algorithms.

Although all algorithms in this paper are recognition algorithms, they can be easily extended to become parsing algorithms, and therefore we speak of “gap parsing” rather than “gap recognition”.

This paper is structured as follows. We summarize some standard definitions and results in Section 2, and briefly discuss Earley’s algorithm in Section 3. Sections 4 through 7 study upper bounds on the time complexities of different types of gap parsing. Section 8 discusses extensions of these results to strong LL( $k$ ) grammars with  $k > 1$  and to XML grammars.

## 2 Preliminaries

A *context-free grammar* (CFG)  $\mathcal{G}$  is a 4-tuple  $(\Sigma, N, S, R)$ , where  $\Sigma$  and  $N$  are two finite disjoint sets of *terminals* and *nonterminals*, respectively,  $S \in N$  is the *start symbol*, and  $R$  is a finite set of *rules*, each of the form  $A \rightarrow \alpha$ , where  $A \in N$  and  $\alpha \in (\Sigma \cup N)^*$ .

For a CFG  $\mathcal{G} = (\Sigma, N, S, R)$ , we define the binary relation  $\Rightarrow_{\mathcal{G}}$  as:  $\alpha \Rightarrow_{\mathcal{G}} \beta$  if and only if  $\alpha$  is of the form  $\gamma A \delta$  and  $\beta$  is of the form  $\gamma \eta \delta$ , for some  $\gamma, \delta \in (\Sigma \cup N)^*$  and a rule  $(A \rightarrow \eta) \in R$ . The reflexive and transitive closure of  $\Rightarrow_{\mathcal{G}}$  is denoted by  $\Rightarrow_{\mathcal{G}}^*$ . We omit subscript  $\mathcal{G}$  when the grammar is understood.

We say a CFG is *reduced* if for each nonterminal  $A$  there are  $\alpha, \beta \in (\Sigma \cup N)^*$  and  $w \in \Sigma^*$  such that  $S \Rightarrow^* \alpha A \beta \Rightarrow^* w$ . We will implicitly assume that all CFGs are reduced. Note that any CFG can be turned into an equivalent CFG that is reduced

[1]. We will in particular use the property of reduced CFGs that for each  $\alpha \in (\Sigma \cup N)^*$  there is a string  $w \in \Sigma^*$  such that  $\alpha \Rightarrow^* w$ .

For each rule  $(A \rightarrow \alpha) \in R$ , we define the lookahead set  $\mathcal{LA}_{\mathcal{G}}(A \rightarrow \alpha)$  as:

$$\{a \in \Sigma \cup \{\#\} \mid \exists v \in \Sigma^*, \gamma \in (\Sigma \cup N)^*, x \in (\Sigma \cup \{\#\})^* [S\# \Rightarrow^* vA\gamma\# \Rightarrow^* v\alpha\gamma\# \Rightarrow^* vax]\}$$

where  $\#$  is an arbitrary symbol not in  $\Sigma$ . We say a grammar  $\mathcal{G}$  is LL(1) if  $\mathcal{LA}_{\mathcal{G}}(A \rightarrow \alpha)$  and  $\mathcal{LA}_{\mathcal{G}}(A \rightarrow \beta)$  are disjoint for each pair of distinct rules  $A \rightarrow \alpha$  and  $A \rightarrow \beta$ . See also [17].

The behaviour of an LL(1) parser for grammar  $\mathcal{G}$  can be formalized on the basis of  $\vdash_{\mathcal{G}}$  defined as the smallest binary relation on  $(\Sigma \cup N)^* \times (\Sigma \cup \{\#\})^*$  such that:

- $(A\beta, aw) \vdash_{\mathcal{G}} (\alpha\beta, aw)$  for each  $(A \rightarrow \alpha) \in R$ ,  $\beta \in (\Sigma \cup N)^*$ ,  $a \in \mathcal{LA}_{\mathcal{G}}(A \rightarrow \alpha)$ , and  $w \in \Sigma^*$ ;
- $(a\beta, aw) \vdash_{\mathcal{G}} (\beta, w)$  for each  $a \in \Sigma$ ,  $\beta \in (\Sigma \cup N)^*$ , and  $w \in \Sigma^*$ .

Again subscript  $\mathcal{G}$  is omitted when the grammar is understood. Note that  $(A, wa) \vdash^* (\epsilon, a)$  implies  $A \Rightarrow^* w$ . (Here  $\epsilon$  denotes the empty string.) Conversely,  $A \Rightarrow^* w$  implies that  $(A, wa) \vdash^* (\epsilon, a)$  for at least one  $a \in \Sigma \cup \{\#\}$ , since we assume the grammar is reduced (choose  $a$  such that  $S\# \Rightarrow^* \alpha A a \beta$ , for some  $\alpha$  and  $\beta$ ). It is well known that  $\mathcal{G}$  is LL(1) if and only if relation  $\vdash_{\mathcal{G}}$  is *deterministic*, by which we mean that for each  $(\alpha, w)$  there is at most one  $(\beta, v)$  such that  $(\alpha, w) \vdash_{\mathcal{G}} (\beta, v)$ ; see [17]. From this fact, it is straightforward to prove:

**Lemma 1** *Assume  $\mathcal{G}$  is LL(1). There are no  $\alpha \in (\Sigma \cup N)^*$ ,  $a, b \in \Sigma \cup \{\#\}$ ,  $w, v \in \Sigma^*$  such that both  $(\alpha, wa) \vdash^* (\epsilon, a)$  and  $(\alpha, wavb) \vdash^* (\epsilon, b)$ .*

For a string  $\alpha \in (\Sigma \cup N)^*$ , its reversal, or mirror image, is denoted by  $\bar{\alpha}$ . The reversal of a CFG  $\mathcal{G} = (\Sigma, N, S, R)$  is a CFG  $\bar{\mathcal{G}} = (\Sigma, N, S, \bar{R})$ , where  $\bar{R}$  consists of the set of rules of the form  $A \rightarrow \bar{\alpha}$  such that  $A \rightarrow \alpha$  is in  $R$ . A CFG  $\mathcal{G}$  is said to be reverse-LL(1) if its reversal  $\bar{\mathcal{G}}$  is LL(1). A CFG  $\mathcal{G}$  is said to be bi-LL(1) if it is both LL(1) and reverse-LL(1).

As these definitions already suggest, an LL(1) grammar is in general not reverse-LL(1), and in fact, if there is an LL(1) grammar describing a language  $L$ , there need not be a reverse-LL(1) grammar describing  $L$ , as exemplified by:

$$L = \{w\bar{c}\bar{w}v \mid w, v \in \{a, b\}^*\}$$

### 3 Earley's algorithm

Earley's algorithm with one symbol of lookahead at the predictor step [9, 6] is presented in Figure 1(a). It is given in an abstract way in the form of a *deduction system*, with the common interpretation as a dynamic programming algorithm [16, 12]. The deduction system consists of a collection of inference rules, each consisting of a list of *antecedents*, which stand for parse items that we have already established to be in a parse table  $\mathcal{T}$ , and, below a horizontal line, the *consequent*, which stands for a

parse item that we derive from the antecedents and that is added to the parse table  $\mathcal{T}$  unless it is already present.

At the right of an inference rule, we may also write a number of *side conditions*, which need to be fulfilled for the inference rule to be applicable. The side conditions refer to rules from a CFG  $\mathcal{G} = (\Sigma, N, S, R)$ , and to an input string  $a_1 \cdots a_q$ . In the following sections of the present article, this string may be a prefix of a longer string  $a_1 \cdots a_q a_{q+1} \cdots a_n$ .

Parse items have the form  $l_r(A \rightarrow \alpha \bullet \beta, i, j)$ , for  $0 \leq i \leq j \leq q$ , where  $l_r$  stands for ‘left to right’. The numbers  $i$  and  $j$  will be called *positions*, and can best be conceived as located between consecutive input positions: position 1 is located between  $a_1$  and  $a_2$ , position 2 between  $a_2$  and  $a_3$ , etc.; position 0 is located before  $a_1$  and position  $q$  after  $a_q$ . If an item  $l_r(A \rightarrow \alpha \bullet \beta, i, j)$  is added to  $\mathcal{T}$ , this implies that:

1.  $S \Rightarrow^* a_1 \cdots a_i A \gamma$ , for some  $\gamma$ , and
2.  $\alpha \Rightarrow^* a_{i+1} \cdots a_j$ .

This is a necessary *and* sufficient condition if  $j = q$ . For  $j < q$ , we need to consider the fact that the algorithm uses lookahead, and a necessary and sufficient condition for  $l_r(A \rightarrow \alpha \bullet \beta, i, j)$  to be added to  $\mathcal{T}$  is that  $(S, a_1 \cdots a_i a_{i+1}) \vdash^* (A \gamma, a_{i+1})$  and  $(\alpha, a_{i+1} \cdots a_j a_{j+1}) \vdash^* (\epsilon, a_{j+1})$  for some  $\gamma$ . If  $a_q = \#$ , then after completion of Earley’s algorithm, there is an item  $l_r(S \rightarrow \alpha \bullet, 0, q-1) \in \mathcal{T}$ , for some  $(S \rightarrow \alpha) \in R$ , if and only if  $a_1 \cdots a_{q-1}$  is in the language generated by  $\mathcal{G}$ .

Figure 1(b) presents Earley’s algorithm in reverse. The side conditions refer to an input string  $a_{q+1} \dots a_n$ , which is possibly a suffix of a longer string  $a_1 \dots a_q a_{q+1} \dots a_n$ . If an item  $r_l(A \rightarrow \alpha \bullet \beta, i, j)$  is added to  $\mathcal{T}$  by this algorithm, this implies that:

1.  $S \Rightarrow^* \gamma A a_{j+1} \cdots a_n$ , for some  $\gamma$ , and
2.  $\beta \Rightarrow^* a_{i+1} \cdots a_j$ .

We interpret Figure 1(a) as a dynamic programming algorithm using a queue  $Q$ , which is to contain derived parse items that still have to be added to  $\mathcal{T}$  and matched against antecedents. At first,  $Q$  contains just  $l_r(S \rightarrow \bullet \alpha, 0, 0)$ , for each rule  $S \rightarrow \alpha$  for which the side conditions in the initializer (1) are satisfied. Then, the following steps are performed until  $Q$  is empty:

1. Remove a parse item  $\Phi$  from  $Q$ .
2. Add  $\Phi$  to  $\mathcal{T}$ .
3. If  $\Phi$  matches the antecedent of the scanner (2) and if the side conditions are satisfied, add the resulting parse item  $\Phi'$  from the consequent to  $Q$  provided  $\Phi'$  occurs neither in  $\mathcal{T}$  nor in  $Q$ .
4. Do the same for the predictor (3).

$$\frac{}{l_r(S \rightarrow \bullet \alpha, 0, 0)} \begin{cases} (S \rightarrow \alpha) \in R \\ 0 = q \vee \\ a_1 \in \mathcal{L}\mathcal{A}_G(S \rightarrow \alpha) \end{cases} \quad (1)$$

$$\frac{l_r(A \rightarrow \alpha \bullet a_{j+1} \beta, i, j)}{l_r(A \rightarrow \alpha a_{j+1} \bullet \beta, i, j + 1)} \{j < q\} \quad (2)$$

$$\frac{l_r(A \rightarrow \alpha \bullet B \beta, i, j)}{l_r(B \rightarrow \bullet \gamma, j, j)} \begin{cases} j \leq q \\ (B \rightarrow \gamma) \in R \\ j = q \vee \\ a_{j+1} \in \mathcal{L}\mathcal{A}_G(B \rightarrow \gamma) \end{cases} \quad (3)$$

$$\frac{\frac{l_r(A \rightarrow \alpha \bullet B \beta, i, j)}{l_r(B \rightarrow \gamma \bullet, j, k)}}{l_r(A \rightarrow \alpha B \bullet \beta, i, k)} \quad (4)$$

(a)

$$\frac{}{r_l(S \rightarrow \alpha \bullet, n, n)} \begin{cases} (S \rightarrow \alpha) \in R \\ q = n \vee \\ a_n \in \mathcal{L}\mathcal{A}_{\bar{G}}(S \rightarrow \alpha) \end{cases} \quad (5)$$

$$\frac{r_l(A \rightarrow \alpha a_i \bullet \beta, i, j)}{r_l(A \rightarrow \alpha \bullet a_i \beta, i - 1, j)} \{q < i\} \quad (6)$$

$$\frac{r_l(A \rightarrow \alpha B \bullet \beta, i, j)}{r_l(B \rightarrow \gamma \bullet, i, i)} \begin{cases} q \leq i \\ (B \rightarrow \gamma) \in R \\ q = i \vee \\ a_i \in \mathcal{L}\mathcal{A}_{\bar{G}}(B \rightarrow \gamma) \end{cases} \quad (7)$$

$$\frac{\frac{r_l(A \rightarrow \alpha B \bullet \beta, j, k)}{r_l(B \rightarrow \gamma \bullet, i, j)}}{r_l(A \rightarrow \alpha \bullet B \beta, i, k)} \quad (8)$$

(b)

Figure 1: Earley's algorithm (a) working on input  $a_1 \cdots a_q$  and its reversed form (b) working on input  $a_{q+1} \cdots a_n$ . Rules (1) - (4) are commonly called initializer, scanner, predictor and completer, respectively.

- 5a.** If  $\Phi$  matches the first antecedent of the completer (4), find matching parse items in  $\mathcal{T}$  for the second antecedent, and add the resulting parse items from the consequent to  $Q$  provided they occur neither in  $\mathcal{T}$  nor in  $Q$ .
- 5b.** If  $\Phi$  matches the second antecedent of the completer (4), find matching parse items in  $\mathcal{T}$  for the first antecedent, and add the resulting parse items from the consequent to  $Q$  provided they occur neither in  $\mathcal{T}$  nor in  $Q$ .

Our subsequent analysis of the time complexity relies on the assumption that parse items stored in  $\mathcal{T}$  are indexed by each input position separately, and by both input positions together. Also parse items in  $Q$  are indexed by both input positions. For step (5a) above, if a parse item of the form  $l_r(A \rightarrow \alpha \bullet B\beta, i, j)$  is taken from the queue, then all matching parse items  $l_r(B \rightarrow \gamma \bullet, j, k)$  can be found in unit time per item, as we assume that the latter items are indexed by  $j$ . As each parse item is retrieved at most once from  $Q$ , the time costs of step (5a) are proportional to the number of combinations of items of the form  $l_r(A \rightarrow \alpha \bullet B\beta, i, j)$  with items of the form  $l_r(B \rightarrow \gamma \bullet, j, k)$ . A similar consideration holds for step (5b). Simpler arguments can be given for steps (3) and (4). As parse items are also indexed by combinations of the two input positions, we may check in unit time whether a newly derived parse item is already present in  $\mathcal{T}$  and whether it is already present in  $Q$ .

We conclude that each inference rule can be implemented in such a way that the number of steps it contributes to the overall time costs is proportional to the number of valid instantiations of its list of antecedents. This holds in general for inference rules with up to two antecedents. We will discuss an inference rule with three antecedents below; it requires an appropriate refinement of the above analysis.

In the presence of an LL(1) grammar, Earley's algorithm behaves very similarly to a usual stack-based LL(1) parser, and therefore it is not unexpected that the time complexity is linear. We first prove:

**Lemma 2** *Assume  $\mathcal{G}$  is LL(1). After completion of Earley's algorithm (Figure 1(a)), there can be at most one item  $l_r(A \rightarrow \alpha \bullet \beta, i, j) \in \mathcal{T}$  for each  $i$  and  $A \rightarrow \alpha \bullet \beta$ .*

Suppose that after completion of Earley's algorithm there are two different  $j_1$  and  $j_2$  (say  $j_1 < j_2$ ) such that  $l_r(A \rightarrow \alpha \bullet \beta, i, j_1), l_r(A \rightarrow \alpha \bullet \beta, i, j_2) \in \mathcal{T}$  for some combination of  $i$  and  $A \rightarrow \alpha \bullet \beta$ . Because  $j_1 < q$ , lookahead is used and we have  $(\alpha, a_{i+1} \cdots a_{j_1} a_{j_1+1}) \vdash^* (\epsilon, a_{j_1+1})$ , but also  $(\alpha, a_{i+1} \cdots a_{j_1} a_{j_1+1} \cdots a_{j_2} b) \vdash^* (\epsilon, b)$ , some  $b \in \Sigma \cup \{\#\}$ . This is contradicted by Lemma 1, which completes the proof of Lemma 2. ■

**Theorem 3** *Assume  $\mathcal{G}$  is LL(1). The time complexity of Earley's algorithm (Figure 1(a)) is  $\mathcal{O}(q)$ .*

To prove this, it is sufficient to note that each inference rule can be applied for at most  $\mathcal{O}(q)$  distinct combinations of items, which follows directly from Lemma 2. ■

By symmetry, we also have:

**Theorem 4** *Assume  $\mathcal{G}$  is reverse-LL(1). The time complexity of Earley's algorithm in reverse (Figure 1(b)) is  $\mathcal{O}(n - q)$ .*

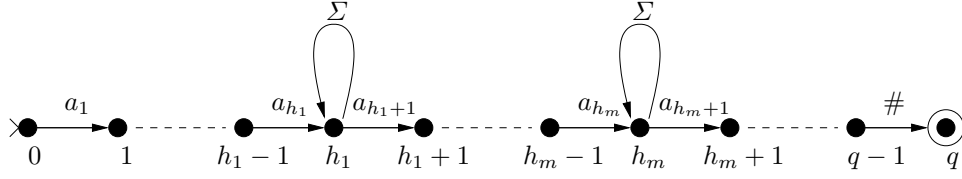


Figure 2: Input  $a_1 \cdots a_q$  with gaps in  $\mathcal{H} = \{h_1, \dots, h_m\}$  represented as finite automaton.

$$\frac{l_r(A \rightarrow \alpha \bullet a\beta, i, j)}{l_r(A \rightarrow \alpha a \bullet \beta, i, j)} \{j \in \mathcal{H}\} \quad (9)$$

$$\frac{l_r(A \rightarrow \alpha \bullet B\beta, i, j)}{l_r(B \rightarrow \bullet \gamma, j, j)} \left\{ \begin{array}{l} j \in \mathcal{H} \\ (B \rightarrow \gamma) \in R \end{array} \right. \quad (10)$$

Figure 3: Two inference rules to be added to Figure 1(a) in order to handle the general problem of gap parsing.

## 4 Gap parsing

Let us now consider the general problem of gap parsing. In addition to an input string  $a_1 \cdots a_q$ , with  $a_q = \#$ , we have a set  $\mathcal{H}$  of positions between 0 and  $q$  where an arbitrary string may be inserted. Formally, let  $\mathcal{H} = \{h_1, \dots, h_m\}$ , where  $0 \leq h_1 < \dots < h_m < q$ . The completed strings are of the form  $a_1 \cdots a_{h_1} w_1 a_{h_1+1} \cdots a_{h_2} w_2 a_{h_2+1} \cdots a_{h_m} w_m a_{h_m+1} \cdots a_q$ , for some  $w_1, \dots, w_m \in \Sigma^*$ . The set of completed strings is accepted by the finite automaton sketched in Figure 2.

The task is now to decide whether there is a completed string that is generated by a given CFG  $\mathcal{G}$ . Earley's algorithm can be adapted to handle this task by adding the two inference rules in Figure 3. These are variants of the familiar scanner and predictor steps, with the only difference that scanning of any string  $w_l$  can be simulated at an input position that is a gap  $h_l \in \mathcal{H}$ , in combination with the existing completer step. The answer to the problem is positive if and only if after completion of the algorithm there is an item  $l_r(S \rightarrow \alpha \bullet, 0, q-1) \in \mathcal{T}$  for some  $(S \rightarrow \alpha) \in R$ . Correctness of the algorithm straightforwardly follows from the above characterization of inference rules (9) and (10).

**Theorem 5** *Let  $\mathcal{G}$  be an arbitrary CFG. The time complexity of Earley's algorithm with arbitrary gaps (Figure 1(a) and Figure 3) is  $\mathcal{O}(q^3)$ .*

This result follows trivially from the observation that all inference rules involve at most 3 input positions, and there are  $\mathcal{O}(q)$  such input positions. ■

Parsing of input strings with gaps has also been studied by [11]. Note the similarity to parsing of finite automata (see Figure 2), which is related to computing the intersection of a context-free and a regular language [2]; see also [19]. Correctness of our gap parsing algorithm follows from the correctness of these well-established algorithms for parsing of finite automata.

## 5 Initial-gap parsing

Now let us consider the case where there is a single gap at the initial position, in other words  $\mathcal{H} = \{0\}$ . Furthermore, we assume that  $\mathcal{G}$  is LL(1).

**Lemma 6** *Assume  $\mathcal{G}$  is LL(1). After completion of Earley's algorithm with initial gap (Figure 1(a) and Figure 3 with  $\mathcal{H} = \{0\}$ ), there can be at most one item  $l_r(A \rightarrow \alpha \bullet \beta, i, j) \in \mathcal{T}$  for each  $i > 0$  and  $A \rightarrow \alpha \bullet \beta$ .*

The proof is identical to that for Lemma 2, except that we exclude the case  $i = 0$ .

■

**Theorem 7** *Assume  $\mathcal{G}$  is LL(1). The time complexity of Earley's algorithm with initial gap is  $\mathcal{O}(q)$ .*

The difference between Lemma 2 and Lemma 6 is that in the case of an initial gap, there can be several items  $l_r(A \rightarrow \alpha \bullet \beta, 0, j) \in \mathcal{T}$  for each  $A \rightarrow \alpha \bullet \beta$ . However, there can obviously be only linearly many of them, and thereby we obtain the same asymptotic complexity as in Theorem 3. ■

## 6 One-gap parsing

Now let us consider the case where there is a single gap at any position, in other words  $\mathcal{H} = \{p\}$ , for some  $p$  with  $0 \leq p < q$ .

**Lemma 8** *Assume  $\mathcal{G}$  is LL(1). After completion of Earley's algorithm with one gap (Figure 1(a) and Figure 3 with  $\mathcal{H} = \{p\}$ ), there can be at most one item  $l_r(A \rightarrow \alpha \bullet \beta, i, j) \in \mathcal{T}$  such that  $j < p$ , for each  $i$  with  $0 \leq i < p$  and  $A \rightarrow \alpha \bullet \beta$ . Furthermore, there can be at most one item  $l_r(A \rightarrow \alpha \bullet \beta, i, j) \in \mathcal{T}$  for each  $i > p$  and  $A \rightarrow \alpha \bullet \beta$ .*

The proof is identical to that for Lemma 2, except that we exclude the case  $i \leq p \leq j$ .

■

**Theorem 9** *Assume  $\mathcal{G}$  is LL(1). The time complexity of Earley's algorithm with one gap is  $\mathcal{O}(q^2)$ .*

For each  $i \leq p$ , there can now be  $\mathcal{O}(q)$  items of the form  $l_r(A \rightarrow \alpha \bullet \beta, i, j) \in \mathcal{T}$  with  $j \geq p$ , which is where Lemma 8 differs from Lemma 2. Theorem 9 follows from the observation that each inference rule can now be applied for at most  $\mathcal{O}(q^2)$  distinct combinations of items, as can be easily verified. ■

$$\frac{l_r(A \rightarrow \alpha \bullet a\beta, i, q)}{l_r\text{-max}(A \rightarrow \alpha \bullet a\beta, i, q)} \quad (11)$$

$$\frac{l_r(A \rightarrow \alpha \bullet B\beta, i, j)}{l_r\text{-max}(B \rightarrow \gamma \bullet \delta, j, k)} \quad (12)$$

$$\frac{r_l(A \rightarrow \alpha a \bullet \beta, q, j)}{r_l\text{-max}(A \rightarrow \alpha a \bullet \beta, q, j)} \quad (13)$$

$$\frac{r_l(A \rightarrow \alpha B \bullet \beta, j, k)}{r_l\text{-max}(B \rightarrow \gamma \bullet \delta, i, j)} \quad (14)$$

$$\frac{l_r\text{-max}(S \rightarrow \alpha \bullet \beta\gamma, 0, i)}{r_l\text{-max}(S \rightarrow \alpha\beta \bullet \gamma, j, n)} \quad (15)$$

$$\frac{\text{bridge}(A \rightarrow \alpha \bullet B \bullet \beta, i, j)}{l_r\text{-max}(B \rightarrow \gamma \bullet \delta\eta, i, i')} \quad (16)$$

$$\frac{\text{bridge}(A \rightarrow \alpha \bullet \beta \bullet \gamma, i, j)}{\text{success}} \quad \{|\beta| > 1\} \quad (17)$$

$$\frac{\text{bridge}(A \rightarrow \alpha \bullet \beta \bullet \gamma, q, q)}{\text{success}} \quad (18)$$

Figure 4: Additional inference rules to handle a single gap at position  $q$ , in combination with the inference rules in Figure 1.

Let us now assume that  $\mathcal{G}$  is bi-LL(1). As we will show, with this further restriction on  $\mathcal{G}$  we may improve the time complexity of one-gap parsing from quadratic to linear. This relies on applying Earley's algorithm in two directions, as follows. Let the input be  $a_1 \cdots a_q a_{q+1} \cdots a_n$ , with the gap at position  $q$ . Earley's algorithm is used to process the prefix  $a_1 \cdots a_q$  by means of the inference rules in Figure 1(a). The reversed algorithm is used to process suffix  $a_{q+1} \cdots a_n$  by means of the inference rules in Figure 1(b).

We also need the additional inference rules in Figure 4 to connect the left-to-right and right-to-left items. The intuition behind the items derived by inference rules (11)–(14) of Figure 4 is that we mark items corresponding to rule occurrences where recognition cannot be continued because of the gap. Formally, we obtain an item  $l_r\text{-max}(A \rightarrow \alpha \bullet X\beta, i, j) \in \mathcal{T}$  next to an item  $l_r(A \rightarrow \alpha \bullet X\beta, i, j) \in \mathcal{T}$ , with  $X \in \Sigma \cup N$ , if and only if:

1.  $S \Rightarrow^* a_1 \cdots a_i A \gamma$ , for some  $\gamma$ ,
2.  $A \rightarrow \alpha X \beta$  is in  $R$ ,
3.  $\alpha \Rightarrow^* a_{i+1} \cdots a_j$ , and
4.  $X \Rightarrow^* a_{j+1} \cdots a_q v$ , for some  $v \neq \epsilon$ .

Similarly, we obtain an item  $r_l\text{-max}(A \rightarrow \alpha X \bullet \beta, i, j) \in \mathcal{T}$ , if and only if:

1.  $S \Rightarrow^* \gamma A a_{j+1} \cdots a_n$ , for some  $\gamma$ ,

2.  $A \rightarrow \alpha X \beta$  is in  $R$ ,
3.  $\beta \Rightarrow^* a_{i+1} \cdots a_j$ , and
4.  $X \Rightarrow^* w a_{q+1} \cdots a_i$ , for some  $w \neq \epsilon$ .

The items of the form  $bridge(A \rightarrow \alpha \bullet \beta \bullet \gamma, i, j)$ , derived by inference rules (15)–(16) in Figure 4, serve to connect the two directions of processing, in a top-down manner. Note that in such items we always have  $i \leq q \leq j$ . Formally, we obtain an item  $bridge(A \rightarrow \alpha \bullet X_1 \beta X_2 \bullet \gamma, i, j) \in \mathcal{T}$  if and only if for some  $i'$  and  $j'$  ( $i \leq i' \leq j' \leq j$ ):

1.  $S \Rightarrow^* a_1 \cdots a_{i'} A a_{j'+1} \cdots a_n$ ,
2.  $A \rightarrow \alpha X_1 \beta X_2 \gamma$  is in  $R$ ,
3.  $\alpha \Rightarrow^* a_{i'+1} \cdots a_i$ ,
4.  $\gamma \Rightarrow^* a_{j+1} \cdots a_{j'}$ ,
5.  $X_1 \Rightarrow^* a_{i+1} \cdots a_q v$ , for some  $v \neq \epsilon$ , and
6.  $X_2 \Rightarrow^* w a_{q+1} \cdots a_j$ , for some  $w \neq \epsilon$ .

An item  $bridge(A \rightarrow \alpha \bullet X \bullet \gamma, i, j) \in \mathcal{T}$  is obtained under similar conditions, replacing both  $X_1$  and  $X_2$  in the above by  $X$  and  $A \rightarrow \alpha X_1 \beta X_2 \gamma$  by  $A \rightarrow \alpha X \gamma$ .

The inference rules (17)–(18) in Figure 4 explicitly specify when parsing is successful, i.e. when we have found an item that implies that  $a_1 \cdots a_q x a_{q+1} \cdots a_n$  is generated by  $\mathcal{G}$  for some  $x \in \Sigma^*$ . It can be easily justified that an item  $bridge(A \rightarrow \alpha \bullet X_1 \beta X_2 \bullet \gamma, i, j) \in \mathcal{T}$ , with the characterization as above, signals that the input should be recognized, since if  $X_1 \Rightarrow^* a_{i+1} \cdots a_q v$ ,  $X_2 \Rightarrow^* w a_{q+1} \cdots a_j$  and  $y \in \Sigma^*$  is an arbitrary string such that  $\beta \Rightarrow^* y$ , then we may take  $x = v y w$  as filling of the gap. Similarly, for an item  $bridge(A \rightarrow \alpha \bullet \beta \bullet \gamma, q, q) \in \mathcal{T}$  we may take any string  $x \in \Sigma^*$  such that  $\beta \Rightarrow^* x$  as filling of the gap. Conversely, if there is some  $x \in \Sigma^*$  such that  $a_1 \cdots a_q x a_{q+1} \cdots a_n$  is generated by  $\mathcal{G}$ , then we will be able to derive  $success \in \mathcal{T}$ .

**Lemma 10** *Assume  $\mathcal{G}$  is bi-LL(1). After completion of the bidirectional one-gap algorithm (Figure 1 and Figure 4), there can be at most one item  $l\_r\_max(A \rightarrow \alpha \bullet \beta, i, j) \in \mathcal{T}$  such that  $j < q$ , for each  $i < q$  and  $A$ , and at most one item  $r\_l\_max(A \rightarrow \alpha \bullet \beta, i, j) \in \mathcal{T}$  such that  $q < i$ , for each  $j > q$  and  $A$ .*

Suppose there are two distinct items  $l\_r\_max(A \rightarrow \alpha_1 \bullet \beta_1, i, j_1) \in \mathcal{T}$  and  $l\_r\_max(A \rightarrow \alpha_2 \bullet \beta_2, i, j_2) \in \mathcal{T}$ , for  $i, j_1$  and  $j_2$  such that  $i \leq j_1 \leq j_2 < q$ . The possibility of  $\alpha_1 \beta_1 \neq \alpha_2 \beta_2$  is excluded by the assumption that  $\mathcal{G}$  is LL(1), and due to the use of lookahead in the algorithm. It follows that  $j_1 < j_2$  and the two items can be written as  $l\_r\_max(A \rightarrow \alpha \bullet X \beta \gamma, i, j_1) \in \mathcal{T}$  and  $l\_r\_max(A \rightarrow \alpha X \beta \bullet \gamma, i, j_2) \in \mathcal{T}$ . Therefore there must be two items  $l\_r(A \rightarrow \alpha \bullet X \beta \gamma, i, j_1) \in \mathcal{T}$

and  $l_r(A \rightarrow \alpha X \beta \bullet \gamma, i, j_2) \in \mathcal{T}$ , and due to the nature of the algorithm, especially Lemma 2, there must also be an item  $l_r(A \rightarrow \alpha X \bullet \beta \gamma, i, j_3) \in \mathcal{T}$ , for some  $j_3 \leq j_2$ , and  $(X, a_{j_1+1} \cdots a_{j_3} a_{j_3+1}) \vdash^* (\epsilon, a_{j_3+1})$ . Furthermore,  $l_r\text{-max}(A \rightarrow \alpha \bullet X \beta \gamma, i, j_1) \in \mathcal{T}$  implies  $X \Rightarrow^* a_{j_1+1} \cdots a_{j_3} a_{j_3+1} \cdots a_q v$ , for some  $v \neq \epsilon$ , and thereby  $(X, a_{j_1+1} \cdots a_{j_3} a_{j_3+1} \cdots a_q v b) \vdash^* (\epsilon, b)$ , for some  $b \in \Sigma \cup \{\#\}$ . This is contradicted by Lemma 1. A symmetrical discussion of the remaining case concludes the proof of Lemma 10. ■

**Lemma 11** *Assume  $\mathcal{G}$  is bi-LL(1). After completion of the bidirectional one-gap algorithm, there are  $\mathcal{O}(n)$  items of the form  $\text{bridge}(A \rightarrow \alpha \bullet \beta \bullet \gamma, i, j) \in \mathcal{T}$ .*

Due to Lemma 10, inference rule (15) can derive at most one item. Due to the same lemma, inference rule (16) allows us to derive at most one item  $\text{bridge}(B \rightarrow \gamma \bullet \delta \bullet \eta, i', j') \in \mathcal{T}$  with  $i' < q < j'$  on the basis of an item  $\text{bridge}(A \rightarrow \alpha \bullet B \bullet \beta, i, j) \in \mathcal{T}$ , and  $i \leq i' \leq j' \leq j$ . Note that the number of distinct  $A \rightarrow \alpha \bullet \beta \bullet \gamma$  is a constant determined by the grammar. Furthermore, there are  $\mathcal{O}(n)$  items of the form  $\text{bridge}(B \rightarrow \gamma \bullet \delta \bullet \eta, i', j') \in \mathcal{T}$  with  $i' = q$  or  $j' = q'$ . This concludes the result of Lemma 11. ■

**Theorem 12** *Assume  $\mathcal{G}$  is bi-LL(1). The time complexity of the bidirectional one-gap algorithm is  $\mathcal{O}(n)$ .*

The inference rules in Figure 1 and the inference rules (11)–(14) of Figure 4 can be applied  $\mathcal{O}(n)$  times, which can be proven analogously to the proofs of Theorem 3 and Theorem 4. Due to Lemma 10 and Lemma 11, the inference rules (15)–(16) can be applied  $\mathcal{O}(n)$  times, and naturally the same holds for inference rules (17)–(18).

As (16) has three antecedents, our earlier analysis of the time complexity of deduction systems is no longer adequate. We can however refine our analysis to include inference rules with more than two antecedents. Following [12], we observe that an inference rule such as (16) can be decomposed into a collection of rules with only two antecedents each, by introducing intermediate parse items:

$$\frac{\text{bridge}(A \rightarrow \alpha \bullet B \bullet \beta, i, j) \quad l_r\text{-max}(B \rightarrow \gamma \bullet \delta \eta, i, i')}{\text{intermediate}(B \rightarrow \gamma \bullet \delta \eta, i', j)} \quad \frac{\text{intermediate}(B \rightarrow \gamma \bullet \delta \eta, i', j) \quad r_l\text{-max}(B \rightarrow \gamma \delta \bullet \eta, j', j)}{\text{bridge}(B \rightarrow \gamma \bullet \delta \bullet \eta, i', j')}$$

Clearly, only  $\mathcal{O}(n)$  such intermediate parse items can be derived. By the familiar reasoning for inference rules with at most two antecedents, the proof is now complete. ■

There are superficial similarities between our items  $\text{bridge}(A \rightarrow \alpha \bullet \beta \bullet \gamma, i, j)$  and the items used in some existing algorithms of bidirectional parsing, e.g. [8], in that both types of items possess two distinguished positions in the right-hand side of a grammar rule. However, in the existing algorithms, such an item indicates that  $\beta$  has been matched to input between positions  $i$  and  $j$ , whereas in the present algorithm it indicates that  $\alpha$  and  $\gamma$  have been matched to input before position  $i$  and after position  $j$ , respectively.

Figure 5 illustrates application of the bidirectional one-gap algorithm.

$\mathcal{G}$  contains rules:

$$\begin{aligned} S &\rightarrow a S a \\ S &\rightarrow b S b \\ S &\rightarrow c \end{aligned}$$

Input with positions:

$$\begin{array}{cccccc} & a & b & b & b & a \\ 0 & 1 & 2 & 3 & 4 & 5 \end{array}$$

Computed items:

$$\begin{array}{ll} l_r(S \rightarrow \bullet a S a, 0, 0) & r_l(S \rightarrow a S a \bullet, 5, 5) \\ l_r(S \rightarrow a \bullet S a, 0, 1) & r_l(S \rightarrow a S \bullet a, 4, 5) \\ l_r(S \rightarrow \bullet b S b, 1, 1) & r_l(S \rightarrow b S b \bullet, 4, 4) \\ l_r(S \rightarrow b \bullet S b, 1, 2) & r_l(S \rightarrow b S \bullet b, 3, 4) \\ l_r(S \rightarrow \bullet b S b, 2, 2) & r_l(S \rightarrow a S a \bullet, 3, 3) \\ l_r(S \rightarrow b \bullet S b, 2, 3) & r_l(S \rightarrow b S b \bullet, 3, 3) \\ l_r(S \rightarrow \bullet a S a, 3, 3) & r_l(S \rightarrow c \bullet, 3, 3) \\ l_r(S \rightarrow \bullet b S b, 3, 3) & r_{l\_max}(S \rightarrow a S a \bullet, 3, 3) \\ l_r(S \rightarrow \bullet c, 3, 3) & r_{l\_max}(S \rightarrow b S b \bullet, 3, 3) \\ l_{r\_max}(S \rightarrow \bullet a S a, 3, 3) & r_{l\_max}(S \rightarrow c \bullet, 3, 3) \\ l_{r\_max}(S \rightarrow \bullet b S b, 3, 3) & r_{l\_max}(S \rightarrow b S \bullet b, 3, 4) \\ l_{r\_max}(S \rightarrow \bullet c, 3, 3) & r_{l\_max}(S \rightarrow a S \bullet a, 4, 5) \\ l_{r\_max}(S \rightarrow b \bullet S b, 2, 3) & \\ l_{r\_max}(S \rightarrow b \bullet S b, 1, 2) & \\ l_{r\_max}(S \rightarrow a \bullet S a, 0, 1) & \\ & bridge(S \rightarrow a \bullet S \bullet a, 1, 4) \\ & bridge(S \rightarrow b \bullet S \bullet b, 2, 3) \\ & bridge(S \rightarrow b \bullet S b \bullet, 3, 3) \\ & success \end{array}$$

Figure 5: Application of the bidirectional one-gap algorithm, for the bi-LL(1) grammar above, generating palindromes over  $\{a, b\}$  with center marker  $c$ , and input  $abbba$  with a gap at position 3.

## 7 Two-gap parsing

Now let us consider the case where there are two gaps in string  $a_1 \cdots a_n$ , at positions  $p$  and  $q$ , where  $0 \leq p < q \leq n$ . We assume that  $\mathcal{G}$  is bi-LL(1). We can now solve the parsing problem by combining the two algorithms from the previous section. We use Figure 1 and Figure 4 to handle the gap at position  $q$ , which we combine with Figure 3 to handle the gap at position  $p$ , defining  $\mathcal{H} = \{p\}$ . Due to the gap at position  $p$ , there may be several items  $l_r(A \rightarrow \alpha \bullet \beta, i, j) \in \mathcal{T}$  such that  $j \geq p$  for a single combination of  $A \rightarrow \alpha \bullet \beta$  and  $i \leq p$ . Consequently, the number of items involved in processing  $a_1 \cdots a_q$  is now quadratic instead of linear, which leads to an overall quadratic time complexity, as we will show.

**Lemma 13** *Assume  $\mathcal{G}$  is bi-LL(1). After completion of the bidirectional two-gap*

algorithm (Figure 1, Figure 4 and Figure 3 with  $\mathcal{H} = \{p\}$ ), there can be at most one item  $l\_r\_max(A \rightarrow \alpha \bullet \beta, i, j) \in \mathcal{T}$  such that  $j < p$ , for each  $i$  with  $0 \leq i < p$  and each  $A$ , and at most one item  $l\_r\_max(A \rightarrow \alpha \bullet \beta, i, j) \in \mathcal{T}$  such that  $j < q$ , for each  $i > p$  and  $A$ . Furthermore, there can be at most one item  $r\_l\_max(A \rightarrow \alpha \bullet \beta, i, j) \in \mathcal{T}$  such that  $q < i$ , for each  $j > q$  and  $A$ .

The proof is almost identical to that of Lemma 10, except that we exclude the case  $i \leq p \leq j$  for items  $l\_r\_max(A \rightarrow \alpha \bullet \beta, i, j) \in \mathcal{T}$ . ■

**Lemma 14** *Assume  $\mathcal{G}$  is bi-LL(1). After completion of the bidirectional two-gap algorithm, there are  $\mathcal{O}(n)$  items of the form  $bridge(A \rightarrow \alpha \bullet \beta \bullet \gamma, i, j) \in \mathcal{T}$  such that  $i \leq p$ .*

For the case  $i < p$ , the proof is almost identical to that of Lemma 11, making use of Lemma 13. There are trivially also  $\mathcal{O}(n)$  items of the form  $bridge(A \rightarrow \alpha \bullet \beta \bullet \gamma, p, j) \in \mathcal{T}$ . ■

**Theorem 15** *Assume  $\mathcal{G}$  is bi-LL(1). The time complexity of the bidirectional two-gap algorithm is  $\mathcal{O}(n^2)$ .*

The inference rules in Figure 1 and Figure 3 can be applied  $\mathcal{O}(n^2)$  times; cf. Theorem 9 and Theorem 4. Also inference rules (11)–(15) and (17)–(18) of Figure 4 can clearly be applied  $\mathcal{O}(n^2)$  times.

For inference rule (16) we need to distinguish two main cases. First, there are  $\mathcal{O}(n)$  items of the form  $bridge(A \rightarrow \alpha \bullet B \bullet \beta, i, j) \in \mathcal{T}$  such that  $i \leq p$ , due to Lemma 14. For each of these, we can apply inference rule (16) with  $\mathcal{O}(n)$  items  $l\_r\_max(B \rightarrow \gamma \bullet \delta \eta, i, i')$  and at most one item  $r\_l\_max(B \rightarrow \gamma \delta \bullet \eta, j', j)$  with  $j' > q$ , due to Lemma 13, and at most a constant number of such items with  $j' = q$ . (The constant is determined by the grammar.) Second, for each of the  $\mathcal{O}(n^2)$  items of the form  $bridge(A \rightarrow \alpha \bullet B \bullet \beta, i, j) \in \mathcal{T}$  such that  $i > p$ , we can apply inference rule (16) with at most one item  $l\_r\_max(B \rightarrow \gamma \bullet \delta \eta, i, i')$  with  $i' < q$ , due to Lemma 13, and at most a constant number of such items with  $i' = q$ , and at most one item  $r\_l\_max(B \rightarrow \gamma \delta \bullet \eta, j', j)$  with  $j' > q$ , due to Lemma 13, and at most a constant number of such items with  $j' = q$ . With all these cases together, inference rule (16) can be applied  $\mathcal{O}(n^2)$  times. This concludes the proof of Theorem 15. ■

Let us now consider the question whether Theorem 15 can be extended to all LL(1) grammars. The following LL(1) grammar illustrates the hardness of this problem:

$$\begin{aligned} S &\rightarrow A D \\ A &\rightarrow a A B \mid d \\ B &\rightarrow b B C \mid c \\ C &\rightarrow b \mid c \\ D &\rightarrow c D \mid \epsilon \end{aligned}$$

For input  $a^p b^p c^p$  with gaps at  $p$  and at  $2p$ , there may be  $\mathcal{O}(p)$  positions within  $a^p$  where an occurrence of rule  $A \rightarrow a A B$  begins,  $\mathcal{O}(p)$  positions within  $b^p$  for

the beginning of  $B$  within this rule occurrence, and  $\mathcal{O}(p)$  positions within  $c^p$  where the rule occurrence ends. Therefore the time complexity of Earley's algorithm with two gaps (Figure 1(a) and Figure 3 with  $\mathcal{H} = \{p, 2p\}$ ) would not be bounded by a quadratic function on the length of the input string. Similar observations can be made for other parsing algorithms that use some kind of completer step akin to (4). Although there may be quadratic-time *recognition* algorithms for this special case, it seems unlikely that parsing is possible within quadratic time for the full class of LL(1) grammars and input with two gaps.

The next question we address is whether Theorem 15 can be extended to bi-LL(1) grammars and *three* gaps. The following bi-LL(1) grammar illustrates the hardness of this problem:

$$\begin{aligned} A &\rightarrow a A B \mid d \\ B &\rightarrow b B b \mid c \end{aligned}$$

Let us consider input  $a^p b^p b^p b^p$  with gaps at  $p$ ,  $2p$  and  $3p$ . Much as for the previous grammar, an occurrence of rule  $A \rightarrow a A B$  may start at  $\mathcal{O}(p)$  different positions within  $a^p$ , the  $B$  of this rule occurrence may start at  $\mathcal{O}(p)$  different positions within the first substring  $b^p$ , and the rule occurrence may end at  $\mathcal{O}(p)$  different positions within the second substring  $b^p$ . Again, it seems unlikely that parsing is possible within quadratic time for the full class of bi-LL(1) grammars and input with three gaps.

## 8 Other classes of grammars

The above results can be easily generalized to grammars that are strong LL( $k$ ) in one or two directions, with  $k \geq 1$ , by extending the use of lookahead in the inference rules to  $k$  symbols.

XML grammars are defined by [4] to describe sets of correctly parenthesized strings. From the perspective of context-free grammars, an XML grammar can be seen as a 4-tuple  $(\Sigma, N, S, R)$ , where  $N$  is a finite set of nonterminals,  $S \in N$  is the start symbol,  $\Sigma$  is the set of open and close brackets  $[_A$  and  $]_A$  such that  $A \in N$ , and there is precisely one rule  $A \rightarrow [_A L_A ]_A$  in  $R$  for each nonterminal  $A \in N$ , where  $L_A$  is a regular language over  $N$ . Such a rule represents a potentially infinite set of rules of the form  $A \rightarrow [_A \alpha ]_A$ , where  $\alpha \in L_A$ . Consequently, if  $x \in \Sigma^*$  is such that  $A \Rightarrow^* x$ , then  $x$  is a Dyck prime. (A Dyck prime is a balanced sequence of parentheses without a proper prefix having the same property.)

XML grammars and bi-LL(1) grammars are incomparable formalisms, and therefore we cannot reuse the algorithms from the previous sections, but we may devise very similar algorithms for gap parsing. A key property of XML grammars is that each open bracket  $[_A$  uniquely determines a rule  $A \rightarrow [_A L_A ]_A$ , and so does each close bracket  $]_A$ . This is quite similar to the bi-LL(1) property.

For convenience, let us represent the right-hand side of a rule  $A \rightarrow [_A L_A ]_A$  as a finite automaton  $\mathcal{M}_A$ , with alphabet  $\{[_A, ]_A\} \cup N$ , initial state  $q_0$  and final state  $q_f$ , which accepts the set of strings of the form  $[_A \alpha ]_A$  such that  $\alpha \in L_A$ . We may

now introduce items similar to those in Section 3 to parse a prefix or a suffix of a string by a variant of Earley’s algorithm or its reversed form. The left-to-right version manipulates items  $l_r(\mathcal{M}_A, q, i, j)$ , where  $\mathcal{M}_A$  is the finite automaton associated with nonterminal  $A$ , and  $q$  is a state in that automaton. Such an item indicates that there is a path of transitions in automaton  $\mathcal{M}_A$  from the initial state  $q_0$  to  $q$  that scans input string  $\alpha \in (\{[A, ]_A\} \cup N)^*$ , and  $\alpha \Rightarrow^* a_{i+1} \cdots a_j$ . Items of the form  $r_l(\mathcal{M}_A, q, i, j)$  have the symmetrical meaning.

Similarly, we may introduce items of the form  $l_r\text{-max}(\mathcal{M}_A, q, i, j)$  and  $r_l\text{-max}(\mathcal{M}_A, q, i, j)$ , and items of the form  $\text{bridge}(\mathcal{M}_A, q, q', i, j)$  such that there is at least one path of transitions from state  $q$  to state  $q'$  in  $\mathcal{M}_A$ . In practice one would pre-compile a table that indicates for each pair of states  $q, q'$  from the same automaton  $\mathcal{M}_A$  whether there is a path of transitions leading from  $q$  to  $q'$ .

Let us first consider parsing of the input up to position  $q$  without gaps. Since no Dyck prime may be a prefix of another, we cannot obtain more than one item  $l_r(\mathcal{M}_A, q_f, i, j) \in \mathcal{T}$  for each  $A$  and  $i$ . This can be seen as the equivalent of Lemma 2. By the same observation, we can reformulate the lemmas concerning one-gap and two-gap parsing in Section 6 and Section 7 for the case of XML grammars. By these results, it is straightforward to conclude that the complexity results from Theorem 12 and Theorem 15 carry over from bi-LL(1) grammars to XML grammars.

It is an open question however whether the results from Theorem 12 and Theorem 15 carry over to bi-LR(1) grammars, i.e. whether one-gap and two-gap parsing can be performed in linear time and quadratic time, respectively, for context-free grammars that are LR(1) in both directions. There is existing work on bidirectional LR parsing by [15], which does not improve asymptotic complexities however.

## Acknowledgements

We thank the anonymous referees for valuable comments.

## References

- [1] A.V. Aho and J.D. Ullman. *Parsing*, volume 1 of *The Theory of Parsing, Translation and Compiling*. Prentice-Hall, Englewood Cliffs, N.J., 1972.
- [2] Y. Bar-Hillel, M. Perles, and E. Shamir. On formal properties of simple phrase structure grammars. In Y. Bar-Hillel, editor, *Language and Information: Selected Essays on their Theory and Application*, chapter 9, pages 116–150. Addison-Wesley, Reading, Massachusetts, 1964.
- [3] J. Bates and A. Lavie. Recognizing substrings of LR(k) languages in linear time. *ACM Transactions on Programming Languages and Systems*, 16(3):1051–1077, May 1994.
- [4] J. Berstel and L. Boasson. Formal properties of XML grammars and languages. *Acta Informatica*, 38:649–671, 2002.

- [5] E. Bertsch. An asymptotically optimal algorithm for non-correcting  $LL(1)$  error recovery. Bericht nr. 176, Fakultät für Mathematik, Ruhr-Universität Bochum, April 1994.
- [6] M. Bouckaert, A. Pirotte, and M. Snelling. Efficient parsing algorithms for general context-free parsers. *Information Sciences*, 8:1–26, 1975.
- [7] G.V. Cormack. An LR substring parser for noncorrecting syntax error recovery. *SIGPLAN Notices*, 24(7):161–169, 1989.
- [8] J.P.M. de Vreught and H.J. Honig. Slow and fast parallel recognition. In *Proc. of the Second International Workshop on Parsing Technologies*, pages 127–135, Cancun, Mexico, February 1991.
- [9] J. Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102, February 1970.
- [10] D. Grune and C.J.H. Jacobs. *Parsing Techniques, A Practical Guide*. Ellis Horwood, Chichester, England, 1990.
- [11] B. Lang. Parsing incomplete sentences. In *Proc. of the 12th International Conference on Computational Linguistics*, volume 1, pages 365–371, Budapest, August 1988.
- [12] D. McAllester. On the complexity analysis of static analyses. *Journal of the ACM*, 49(4):512–537, 2002.
- [13] M.-J. Nederhof and E. Bertsch. Linear-time suffix parsing for deterministic languages. *Journal of the ACM*, 43(3):524–554, 1996.
- [14] H. Richter. Noncorrecting syntax error recovery. *ACM Transactions on Programming Languages and Systems*, 7(3):478–489, July 1985.
- [15] H. Saito. Bi-directional LR parsing from an anchor word for speech recognition. In *Papers presented to the 13th International Conference on Computational Linguistics*, volume 3, pages 237–242, 1990.
- [16] S.M. Shieber, Y. Schabes, and F.C.N. Pereira. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24:3–36, 1995.
- [17] S. Sippu and E. Soisalon-Soininen. *Parsing Theory, Vol. II: LR(k) and LL(k) Parsing*, volume 20 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1990.
- [18] B. van Deudekom and P. Kooiman. Top-down non-correcting error recovery in LLgen. Report IR-338, Vrije Universiteit Amsterdam, 1993.
- [19] G. van Noord. The intersection of finite state automata and definite clause grammars. In *33rd Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, pages 159–165, Cambridge, Massachusetts, USA, June 1995.