Left-To-Right Parsing and Bilexical Context-Free Grammars

Mark-Jan Nederhof
DFKI
Stuhlsatzenhausweg 3
D-66123 Saarbrücken
Germany
nederhof@dfki.de

Giorgio Satta
Dipartimento di Elettronica e Informatica
Università di Padova
via Gradenigo, 6/A
I-35131 Padova, Italy
satta@dei.unipd.it

Abstract

We compare the asymptotic time complexity of left-to-right and bidirectional parsing techniques for bilexical context-free grammars, a grammar formalism that is an abstraction of language models used in several state-of-the-art real-world parsers. We provide evidence that left-to-right parsing cannot be realised within acceptable time-bounds if the so called correct-prefix property is to be ensured. Our evidence is based on complexity results for the representation of regular languages.

1 Introduction

Traditionally, algorithms for natural language parsing process the input string strictly from left to right. In contrast, several algorithms have been proposed in the literature that process the input in a bidirectional fashion; see (van Noord, 1997; Satta and Stock, 1994) and references therein. The issue of parsing efficiency for left-to-right vs. bidirectional methods has longly been debated. On the basis of experimental results, it has been argued that the choice of the most favourable strategy should depend on the grammar at hand. With respect to grammar formalisms based upon context-free grammars, and when the rules of these formalisms strongly depend on lexical information, (van Noord, 1997) shows that bidirectional strategies are more efficient than left-to-right strategies. This is because bidirectional strategies are most effective in reducing the parsing search space, by activating as early as possible the maximum number of lexical constraints available in the grammar.

In this paper we present mathematical arguments in support of the above empirically motivated thesis. We investigate a class of lexicalized grammars that, in their probabilistic versions, have been widely adopted as language models in state-of-the-art real-world parsers. The size of these grammars usually grows with the square of the size of the working lexicon, and thus can be very large. In these cases, the primary goal in the design of a parsing algorithm is to achieve asymptotic time performance sublinear in the size of the working grammar and independent of the size of the lexicon. These desiderata are met by existing bidirectional algorithms (Alshawi, 1996; Eisner, 1997; Eisner and Satta, 1999). In contrast, we show the following two main results for the asymptotic time performance of left-to-right algorithms satisfying the so called correct-prefix property.

- In case off-line compilation of the working grammar is not allowed, left-to-right parsing cannot be realised within time bounds independent of the size of the lexicon.
- In case polynomial-time, off-line compilation of the working grammar is allowed, left-to-right parsing cannot be realised in polynomial time, and independently of the size of the lexicon, unless a strong conjecture based on complexity results for the representation of regular languages is falsified.

The first result implies that the well known Earley algorithm and related standard parsing techniques that do not require grammar precompilation cannot be directly extended to process the above mentioned grammars (resp. language models) within an acceptable time bound. The second result provides evidence that well known parsing techniques as left-corner parsing, requiring polynomial-time preprocessing of the grammar, also cannot be directly extended to process these formalisms within an acceptable time bound.

The grammar formalisms we investigate are based upon context-free grammars and are called bilexical context-free grammars. Bilexical context-free grammars have been presented in (Eisner and Satta, 1999) as an abstraction of language models that have been adopted in several recent real-world parsers, improving state-of-the-art parsing accuracy (Alshawi, 1996; Eisner, 1996; Charniak, 1997; Collins, 1997). Our results directly transfer to all these language models. In a bilexical context-free grammar, possible arguments of a word are always specified along with possible head words for those arguments. Therefore a bilexical grammar requires the grammar writer to make stipulations about the compatibil-

ity of particular pairs of words in particular roles, something that was not necessarily true of general context-free grammars.

The remainder of this paper is organized as follows. We introduce bilexical context-free grammars in Section 2, and discuss parsing with the correct-prefix property in Section 3. Our results for parsing with on-line and off-line grammar compilation are presented in Sections 4 and 5, respectively. To complete the presentation, Appendix A shows that left-to-right parsing in time independent of the size of the lexicon is indeed possible when an off-line compilation of the working grammar is allowed that has an exponential time complexity.

2 Bilexical context-free grammars

In this section we introduce the grammar formalism we investigate in this paper. This formalism, originally presented in (Eisner and Satta, 1999), is an abstraction of the language models adopted by several state-of-the-art real-world parsers (see Section 1). We specify a non-stochastic version of the formalism, noting that probabilities may be attached to the rewrite rules exactly as in stochastic CFG (Gonzales and Thomason, 1978; Wetherell, 1980). We assume that the reader is familiar with context-free grammars. Here we follow the notation of (Harrison, 1978; Hopcroft and Ullman, 1979).

A context-free grammar (CFG) is a tuple $G = (V_N, V_T, P, S)$, where V_N and V_T are finite, disjoint sets of nonterminal and terminal symbols, respectively, $S \in V_N$ is the start symbol, and P is a finite set of productions having the form $A \rightarrow \alpha$, where $A \in V_N$ and $\alpha \in (V_N \cup V_T)^*$. A "derives" relation, written $\Rightarrow$, is associated with a CFG as usual. We use the reflexive and transitive closure of $\Rightarrow$, written $\Rightarrow^*$, and define $L(G)$ accordingly. The size of a CFG G is defined as $|G| = \Sigma_{(A \rightarrow \alpha) \in P} |A\alpha|$. If every production in P has the form $A \rightarrow BC$ or $A \rightarrow a$, for $A, B, C \in V_N, a \in V_T$, then G is said to be in Chomsky Normal Form (CNF).

A CFG $G = (V_N, V_T, P, S[\$])$ in CNF is called a **bilexical context-free grammar** if there exists a set V_D, called the set of **delexicalized nonterminals**, such that nonterminals from V_N are of the form $A[a]$, consisting of $A \in V_D$ and $a \in V_T$, and every production in P has one of the following two forms:

(i) $A[a] \rightarrow B[b]\ C[c]$, $a \in \{b, c\}$;

(ii) $A[a] \rightarrow a$.

A nonterminal $A[a]$ is said to have terminal symbol a as its **lexical head**. Note that in a parse tree for G, the lexical head of a nonterminal is always "inherited" from some daughter symbol (i.e., from some symbol in the right-hand side of a production). In the sequel, we also refer to the set V_T as the **lexicon** of the grammar.

A bilexical CFG can encode lexically specific preferences in the form of binary relations on lexical items. For instance, one might specify P as to contain the production VP[solve] $\rightarrow$ V[solve] NP[puzzles] but not the production VP[eat] $\rightarrow$ V[eat] NP[puzzles]. This will allow derivation of some VP constituents such as "solve two puzzles", while forbidding "eat two puzzles". See (Eisner and Satta, 1999) for further discussion.

The cost of this expressiveness is a very large grammar. Indeed, we have $|G| = \mathcal{O}(|V_D|^3 \cdot |V_T|^2)$, and in practical applications $|V_T| \gg |V_D| > 1$. Thus, the grammar size is dominated in its growth by the square of the size of the working lexicon. Even if we conveniently group lexical items with distributional similarities into the same category, in practical applications the resulting grammar might have several thousand productions. Parsing strategies that cannot work in sublinear time with respect to the size of the lexicon *and* with respect to the size of the whole input grammar are very inefficient in these cases.

3 Correct-prefix property

So called left-to-right strategies are standardly adopted in algorithms for natural language parsing. Although intuitive, the notion of left-to-right parsing is a concept with no precise mathematical meaning. Note that in fact, in a pathological way, one could read the input string from left-to-right, storing it into some data structure, and then perform syntactic analysis with a non-left-to-right strategy. In this paper we focus on a precise definition of left-to-right parsing, known in the literature as correct-prefix property parsing (Sippu and Soisalon-Soininen, 1990). Several algorithms commonly used in natural language parsing satisfy this property, as for instance Earley's algorithm (Earley, 1970), tabular left-corner and PLR parsing (Nederhof, 1994) and tabular LR parsing (Tomita, 1986).

Let V_T be some alphabet. A generic string over V_T is denoted as $w = a_1 \cdots a_n$, with $n \geq 0$ and $a_i \in V_T$ $(1 \leq i \leq n)$; in case $n = 0$, w equals the empty string ε. For integers i and j with $1 \leq i \leq j \leq n$, we write $w[i, j]$ to denote string $a_i a_{i+1} \cdots a_j$; if $i > j$, we define $w[i, j] = \varepsilon$.

Let $G = (V_N, V_T, P, S)$ be a CFG and let $w = a_1 \cdots a_n$ with $n \geq 0$ be some string over V_T. A **recognizer** for the CFG class is an algorithm R that, on input $\langle G, w \rangle$, decides whether $w \in L(G)$. We say that R satisfies the **correct-prefix property** (CPP) if the following condition holds. Algorithm R processes the input string from left-to-right, "consuming" one symbol a_i at a time. If for some i, $0 \leq i \leq n$, the set of derivations in G having the form $S \Rightarrow^* w[1, i]\gamma$, $\gamma \in (V_N \cup V_T)^*$, is empty, then R rejects and halts, and it does so before consuming symbol a_{i+1}, if $i < n$. In this case, we say that R

has detected an error at position i in w. Note that the above property forces the recognizer to do relevant computation for each terminal symbol that is consumed.

We say that $w[1,i]$ is a **correct-prefix** for a language L if there exists a string z such that $w[1,i]z \in L$. In the natural language parsing literature, the CPP is sometimes defined with the following condition in place of the above. If for some i, $0 \leq i \leq n$, $w[1,i]$ is not a correct prefix for $L(G)$, then R rejects and halts, and it does so before consuming symbol a_{i+1}, if $i < n$. Note that the latter definition asks for a stronger condition, and the two definitions are equivalent only in case the input grammar G is reduced.[1] While the above mentioned parsing algorithms satisfy the former definition of CPP, they do not satisfy the latter. Actually, we are not aware of any practically used parsing algorithm that satisfies the latter definition of CPP.

One needs to distinguish CPP parsing from some well known parsing algorithms in the literature that process symbols in the right-hand sides of each grammar production from left to right, but that do not exhibit any left-to-right dependency between different productions. In particular, processing of the right-hand side of some production may be initiated at some input position without consultation of productions or parts of productions that may have been found to cover parts of the input to the left of that position. These algorithms may also consult input symbols from left to right, but the processing that takes place to the right of some position i does not strictly depend on the processing that has taken place to the left of i. Examples are pure bottom-up methods, such as left-corner parsing without top-down filtering (Wiren, 1987).

Algorithms that do satisfy the CPP make use of some form of top-down prediction. Top-down prediction can be implemented at parse-time as in the case of Earley's algorithm by means of the "predictor" step, or can be precompiled, as in the case of left-corner parsing (Rosenkrantz and Lewis, 1970), by means of the left-corner relation, or as in the case of LR parsers (Sippu and Soisalon-Soininen, 1990), through the closure function used in the construction of LR states.

4 Recognition without precompilation

In this section we consider recognition algorithms that do not require off-line compilation of the input grammar. Among algorithms that satisfy the CPP, the most popular example of a recognizer that does not require grammar precompilation is perhaps Earley's algorithm (Earley, 1970). We show here that methods in this family cannot be extended to work in time independent of the size of the lexicon, in contrast with bidirectional recognition algorithms.

The result presented below rests on the following, quite obvious, assumption. There exists a constant c, depending on the underlying computation model, such that in $k \geq 0$ elementary computation steps any recognizer can only read up to $c \cdot k$ productions from set P. In what follows, and without any loss of generality, we assume $c = 1$. Apart from this assumption, no other restriction is imposed on the representation of the input grammar or on the access to the elements of sets V_N, V_T and P.

Theorem 1 *Let f be any function of two variables defined on natural numbers. No recognizer for bilexical context-free grammars that satisfies the* CPP *can run on input $\langle G, w\rangle$ in an amount of time bounded by $f(|V_D|, |w|)$, where V_D is the set of delexicalized nonterminals of G.*

Proof. Assume the existence of a recognizer R satisfying the CPP and running in $f(|V_D|, |w|)$ steps or less. We show how to derive a contradiction.

Let $q \geq 1$ be an integer. Define a bilexical CFG $G_q = (V_N^q, V_T^q, P^q, A[b_1])$ where V_T^q contains $q+2$ distinct symbols $\{b_1, \ldots, b_{q+2}\}$ and

$$V_N^q = \{A[b_i] \mid 1 \leq i \leq q+1\} \cup \{T[b] \mid b \in V_T^q\},$$

and where set P^q contains all and only the following productions:

(i) $A[b_i] \rightarrow A[b_{i+1}]\ T[b_i], \quad 1 \leq i \leq q;$

(ii) $A[b_{q+1}] \rightarrow T[b_{q+2}]\ T[b_{q+1}];$

(iii) $T[b] \rightarrow b, \quad b \in V_T^q.$

Productions in (i) are called bridging productions. Note that there are q bridging productions in G_q. Also, note that $V_D^q = \{A, T\}$ does not depend on the choice of q. Thus, we will simply write V_D.

Choose $q > \max\{f(|V_D|, 2), 1\}$. On input $\langle G_q, b_{q+2}b_{q+1}\rangle$, R does not detect any error at position 1, that is after having read the first symbol b_{q+2} of the input string. This is because $A[b_1] \Rightarrow^* b_{q+2}\gamma$ with $\gamma = T[b_{q+1}]\,T[b_q]\,T[b_{q-1}] \cdots T[b_1]$ is a valid derivation in G. Since R executes no more than $f(|V_D|, 2)$ steps, from our assumption that reading a production takes unit time it follows that there must be an integer k, $1 \leq k \leq q$, such that bridging production $A[b_k] \rightarrow A[b_{k+1}]\ T[b_k]$ is not read from G_q. Construct then a new grammar G'_q by replacing in G_q the production $A[b_k] \rightarrow A[b_{k+1}]\ T[b_k]$ with the new production $A[b_k] \rightarrow T[b_k]\ A[b_{k+1}]$, leaving everything else unchanged. It follows that, on input $\langle G'_q, b_{q+2}b_{q+1}\rangle$, R behaves exactly as before and does not detect any error at position 1. But this is

[1] A context-free grammar G is reduced if every nonterminal of G can be part of at least one derivation that rewrites the start symbol into some string of terminal symbols.

a contradiction, since there is no derivation in G'_q of the form $A[b_1] \Rightarrow^* b_{q+2}\gamma$, $\gamma \in (V_N \cup V_T)^*$, as can be easily verified. ■

We can use the above result in the comparison of left-to-right and bidirectional recognizers. The recognition of bilexical context-free languages can be carried out by existing bidirectional algorithms in time independent of the size of the lexicon and without any precompilation of the input bilexical grammar. For instance, the algorithms presented in (Eisner and Satta, 1999) allow recognition in time $O(|V_D|^3 |w|^4)$.[2] Theorem 1 states that this time bound cannot be met if we require the CPP and if the input grammar is not precompiled. In the next section, we will consider the possibility that the input grammar is in a precompiled form.

5 Recognition with precompilation

In this section we consider recognition algorithms that satisfy the CPP and allow off-line, polynomial-time compilation of the working grammar. We focus on a class of bilexical context-free grammars where recognition requires the stacking of a number of unresolved lexical dependencies that is proportional to the length of the input string. We provide evidence that the above class of recognizers perform much less efficiently for these grammars than existing bidirectional recognizers.

We assume that the reader is familiar with the notions of deterministic and nondeterministic finite automata. We follow here the notation in (Hopcroft and Ullman, 1979). A nondeterministic finite automaton (FA) is a tuple $M = (Q, \Sigma, \delta, q_0, F)$, where Q and Σ are finite, disjoint sets of state and alphabet symbols, respectively, $q_0 \in Q$ and $F \subseteq Q$ are the initial state and the set of final states, respectively, and δ is a total function mapping $Q \times \Sigma$ to 2^Q, the power-set of Q. Function δ represents the transitions of the automaton. Given a string $w = a_1 \cdots a_n$, $n \geq 0$, an **accepting computation** in M for w is a sequence $q_0, a_1, q_1, a_2, q_2, \ldots, a_n, q_n$ such that $q_i \in \delta(q_{i-1}, a_i)$ for $1 \leq i \leq n$, and $q_n \in F$. The language $L(M)$ is the set of all strings in Σ^* that admit at least one accepting computation in M. The size of M is defined as $|M| = \Sigma_{q \in Q, a \in \Sigma} |\delta(q, a)|$. The automaton M is deterministic if, for every $q \in Q$ and $a \in \Sigma$, we have $|\delta(q, a)| = 1$.

We call **quasi-determinizer** any algorithm A that satisfies the following two conditions:

1. A takes as input a nondeterministic FA $M = (Q, \Sigma, \delta, q_0, F)$ and produces as output a device D_M that, when given a string w as input, decides whether $w \in L(M)$; *and*
2. there exists a polynomial p_A such that every D_M runs in an amount of time bounded by $p_A(|w|)$.

We remark that, given a nondeterministic FA M specified as above, known algorithms allow simulation of M on an input string w in time $O(|M|\,|w|)$ (see for instance (Aho et al., 1974, Thm. 9.5) or (Sippu and Soisalon-Soininen, 1988, Thm. 3.38)). In contrast, a quasi-determinizer produces a device that simulates M in an amount of time independent of the size of M itself.

A standard example of a quasi-determinizer is the so called power-set construction, used to convert a nondeterministic FA into a language-equivalent deterministic FA (see for instance (Hopcroft and Ullman, 1979, Thm. 2.1) or (Sippu and Soisalon-Soininen, 1988, Thm. 3.30)). In fact, there exist constants c and c' such that any deterministic FA can be simulated on input string w in an amount of time bounded by $c|w| + c'$. This requires function δ to be stored as a $|Q| \times |\Sigma|$, 2-dimensional array with values in Q. This is a standard representation for automata-like structures; see (Gusfield, 1997, Sect. 6.5) for discussion.

We now pose the question of the time efficiency of a quasi-determinizer, and consider the amount of time needed in the construction of D_M. In (Meyer and Fisher, 1971; Stearns and Hunt, 1981) it is shown that there exist (infinitely many) nondeterministic FAs with state set Q, such that any language-equivalent deterministic FA must have at least $2^{|Q|}$ states. This means that the power-set construction cannot work in polynomial time in the size of the input FA. Despite of much effort, no algorithm has been found, up to the authors' knowledge, that can simulate a nondeterministic FA on an input string w in linear time in $|w|$ and independently of $|M|$, if only polynomial-time precompilation of M is allowed. Even in case we relax the linear-time restriction and consider recognition of w in polynomial time, for some fixed polynomial, it seems unlikely that the problem can be solved if only polynomial-time precompilation of M is allowed. Furthermore, if we consider precompilation of nondeterministic FAs into "partially determinized" FAs that would allow recognition in polynomial (or even exponential) time in $|w|$, it seems unlikely that the analysis required for this precompilation could consider less than exponentially many combinations of states that may be active at the same time for the original nondeterministic FA. Finally, although more powerful formalisms have been shown to represent *some* regular languages much more succinctly than FAs (Meyer and Fisher, 1971), while allowing polynomial-time parsing, it seem unlikely that this could hold for regular languages in general.

[2] More precisely, the running time for these algorithms is $O(|V_D|^3 |w|^3 \min\{|V_T|, |w|\})$. In cases of practical interest, we always have $|w| < |V_T|$.

Conjecture *There is no quasi-determinizer that works in polynomial time in the size of the input automaton.*

Before turning to our main result, we need to develop some additional machinery. Let $M = (Q, \Sigma, \delta, q_0, F)$ be a nondeterministic FA and let $w = a_1 \cdots a_n \in L(M)$, where $n \geq 0$. Let $q_0, a_1, q_1, \ldots, a_n, q_n$ be an accepting computation for w in M, and choose some symbol $\$ \notin \Sigma$. We can now encode the accepting computation as

$$(\$, q_0)(a_1, q_1) \cdots (a_n, q_n)$$

where we pair alphabet symbols to states, prepending $\$$ to make up for the difference in the number of alphabet symbols and states. We now provide a construction that associates M with a bilexical CFG G_M. Strings in $L(G_M)$ are obtained by pairing strings in $L(M)$ with encodings of their accepting computations (see below for an example).

Definition 1 *Let $M = (Q, \Sigma, \delta, q_0, F)$ be a nondeterministic* FA. *Choose two symbols $\$, \# \notin \Sigma$, and let $\Delta = \{(a, q) \mid a \in \Sigma \cup \{\$\},\ q \in Q\}$. A bilexical* CFG $G_M = (V_N, V_T, P, C[(\$, q_0)])$ *is specified as follows:*

(i) $V_N = \{T[\sigma] \mid \sigma \in V_T\} \cup \{C[\sigma], C'[\sigma] \mid \sigma \in \Delta\}$;

(ii) $V_T = \Delta \cup \Sigma \cup \{\#\}$;

(iii) *P contains all and only the following productions:*

(a) *for each $\sigma \in V_T$,*
$T[\sigma] \rightarrow \sigma$;

(b) *for each $(a, q), (a', q') \in \Delta$ such that $q' \in \delta(q, a')$,*
$C[(a, q)] \rightarrow C'[(a', q')]\ T[(a, q)]$;

(c) *for each $(a, q) \in \Delta$,*
$C'[(a, q)] \rightarrow T[a]\ C[(a, q)]$;

(d) *for each $(a, q) \in \Delta$ such that $q \in F$,*
$C[(a, q)] \rightarrow T[\#]\ T[(a, q)]$.

We give an example of the above construction. Consider an automaton M and a string $w = a_1 a_2 a_3$ such that $w \in L(M)$. Let $(\$, q_0)(a_1, q_1)(a_2, q_2)(a_3, q_3)$ be the encoding of an accepting computation in M for w. Then the string $a_1 a_2 a_3 \#(a_3, q_3)(a_2, q_2)(a_1, q_1)(\$, q_0)$ belongs to $L(G_M)$. The tree depicted in Figure 1 represents a derivation in G_M of such a string.

The following fact will be used below.

Lemma 1 *For each $w \in \Sigma^*$, $w\#$ is a correct-prefix for $L(G_M)$ if and only if $w \in L(M)$.*

Outline of the proof. We claim the following fact. For each $k > 0$, $a_1, a_2, \ldots, a_k \in \Sigma$ and $q_0, q_1, \ldots, q_k \in Q$ we have

$$q_i \in \delta(q_{i-1}, a_i), \text{ for all i } (1 \leq i \leq k),$$

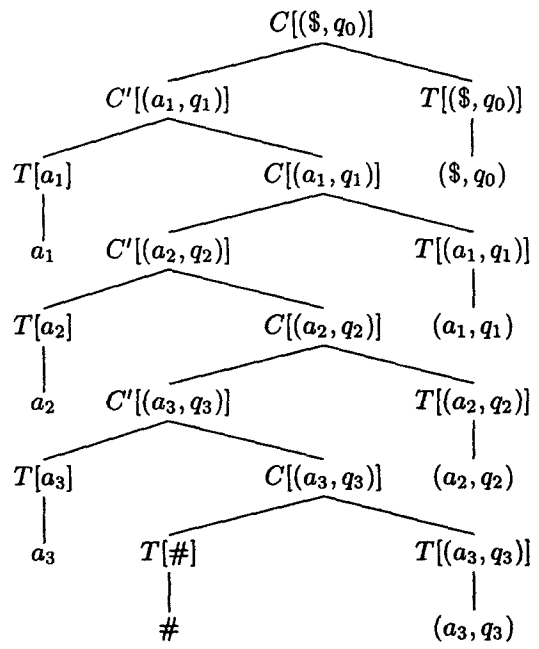

Figure 1: A derivation in G_M for string $a_1 a_2 a_3 \#(a_3, q_3)(a_2, q_2)(a_1, q_1)(\$, q_0)$.

if and only if

$$C[(\$, q_0)] \Rightarrow^* a_1 \cdots a_k C[(a_k, q_k)](a_{k-1}, q_{k-1}) \cdots (\$, q_0).$$

The claim can be proved by induction on k, using productions (a) to (c) from Definition 1.

Let R denote the reverse operator on strings.[3] From the above claim and using production (d) from Definition 1, one can easily show that

$$L(G_M) = \{w\#u \mid w \in L(M),\ u^R \text{ encodes an accepting computation for } w\}.$$

The lemma directly follows from this relation. ∎

We can now provide the main result of this section. To this end, we refine the definition of recognizer presented in Section 3. A recognizer for the CFG class is an algorithm R that has random access to some data structure $C(G)$ obtained by means of some off-line precompilation of a CFG G. On input w, which is a string on the terminal symbols of G, R decides whether $w \in L(G)$. The definition of the CPP extends in the obvious way to recognizers working with precompiled grammars.

Theorem 2 *Let p be any polynomial in two variables. If the conjecture about quasi-determinizers holds true, then no recognizer exists that*

[3] Note that R does not affect individual symbols in a string. Thus $(a, q)^R = (a, q)$.

(i) *has random access to data structure $C(G)$ precompiled from a bilexical CFG G in polynomial time in $|G|$,*

(ii) *runs in an amount of time bounded by $p(|V_D|, |w|)$, where V_D is the set of delexicalized nonterminals of G and w is the input string, and*

(iii) *satisfies the* CPP.

Proof. Assume there exists a recognizer R that satisfies conditions (i) to (iii) in the statement of the theorem. We show how this entails that the conjecture about quasi-determinizers is false.

We use algorithm R to specify a quasi-determinizer A. Given a nondeterministic FA M, A goes through the following steps.

1. A constructs grammar G_M as in Definition 1.
2. A precompiles G_M as required by R, producing data structure $C(G_M)$.
3. A returns a device D_M specified as follows. Given a string w as input, D_M runs R on string $w\#$. If R detects an error at any position i, $0 \leq i \leq |w\#|$, then D_M rejects and halts, otherwise D_M accepts and halts.

From Lemma 1 we have that D_M accepts w if and only if $w \in L(M)$. Since R runs in time $p(|V_D|, |w|)$ and since G_M has a set of delexicalized nonterminals independent of M, we have that there exists a polynomial p_A such that every D_M works in an amount of time bounded by $p_A(|w|)$. We therefore conclude that A is a quasi-determinizer.

It remains to be shown that A works in polynomial time in $|M|$. Step 1 can be carried out in time $\mathcal{O}(|M|)$. The compilation at Step 2 takes polynomial time in $|G_M|$, following our hypotheses on R, and hence polynomial time in $|M|$, since $|G_M| = \mathcal{O}(|M|)$. Finally, the construction of D_M at Step 3 can easily be carried out in time $\mathcal{O}(|M|)$ as well. ∎

In addition to Theorem 1, Theorem 2 states that, even in case the input grammar is compiled off-line and in polynomial time, we cannot perform CPP recognition for bilexical context-free grammars in time polynomial in the grammar and the input string but independent of the lexicon size. This is true with at least the same evidence that supports the conjecture on quasi-determinizers. Again, this should be contrasted with the time performance of existing bidirectional algorithms, allowing recognition for bilexical context-free grammars in time $\mathcal{O}(|V_D|^3 |w|^4)$.

In order to complete our investigation of the above problem, in Appendix A we show that, when we drop the polynomial-time restriction on the grammar precompilation, it is indeed possible to get rid of any $|V_T|$ factor from the running time of the recognizer.

6 Conclusion

Empirical results presented in the literature show that bidirectional parsing strategies can be more time efficient in cases of grammar formalisms whose rules are specialized for one or more lexical items. In this paper we have provided an original mathematical argument in favour of this thesis. Our results hold for bilexical context-free grammars and directly transfer to several language models that can be seen as stochastic versions of this formalism (see Section 1). We perceive that these results can be extended to other language models that properly embed bilexical context-free grammars, as for instance the more general history-based models used in (Ratnaparkhi, 1997) and (Chelba and Jelinek, 1998). We leave this for future work.

Acknowledgements

We would like to thank Jason Eisner and Mehryar Mohri for fruitful discussions. The first author is supported by the German Federal Ministry of Education, Science, Research and Technology (BMBF) in the framework of the VERBMOBIL Project under Grant 01 IV 701 V0, and was employed at AT&T Shannon Laboratory during a part of the period this paper was written. The second author is supported by MURST under project *PRIN: BioInformatica e Ricerca Genomica* and by University of Padua, under project *Sviluppo di Sistemi ad Addestramento Automatico per l'Analisi del Linguaggio Naturale.*

References

A. V. Aho, J. E. Hopcroft, and J. D. Ullman. 1974. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, MA.

H. Alshawi. 1996. Head automata and bilingual tiling: Translation with minimal representations. In *Proc. of the 34th ACL*, pages 167–176, Santa Cruz, CA.

E. Charniak. 1997. Statistical parsing with a context-free grammar and word statistics. In *Proc. of AAAI-97*, Menlo Park, CA.

C. Chelba and F. Jelinek. 1998. Exploiting syntactic structure for language modeling. In *Proc. of the 36th ACL*, Montreal, Canada.

M. Collins. 1997. Three generative, lexicalised models for statistical parsing. In *Proc. of the 35th ACL*, Madrid, Spain.

J. Earley. 1970. An efficient context-free parsing algorithm. *Communications of the Association for Computing Machinery*, 13(2):94–102.

J. Eisner and G. Satta. 1999. Efficient parsing for bilexical context-free grammars and head automaton grammars. In *Proc. of the 37th ACL*, pages 457–464, College Park, Maryland.

J. Eisner. 1996. An empirical comparison of probability models for dependency grammar. Technical Report IRCS-96-11, IRCS, Univ. of Pennsylvania.

J. Eisner. 1997. Bilexical grammars and a cubic-time probabilistic parser. In *Proceedings of the 4th Int. Workshop on Parsing Technologies*, MIT, Cambridge, MA, September.

R. C. Gonzales and M. G. Thomason. 1978. *Syntactic Pattern Recognition.* Addison-Wesley, Reading, MA.

D. Gusfield. 1997. *Algorithms on Strings, Trees and Sequences.* Cambridge University Press, Cambridge, UK.

M. A. Harrison. 1978. *Introduction to Formal Language Theory.* Addison-Wesley, Reading, MA.

J. E. Hopcroft and J. D. Ullman. 1979. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, Reading, MA.

A. R. Meyer and M. J. Fisher. 1971. Economy of description by automata, grammars and formal systems. In *12th Annual Symp. on Switching and Automata Theory*, pages 188–190, New York. IEEE.

M.-J. Nederhof and G. Satta. 1996. Efficient tabular LR parsing. In *Proc. of the 34th ACL*, pages 239–246, Santa Cruz, CA.

M.-J. Nederhof. 1994. An optimal tabular parsing algorithm. In *Proc. of the 32nd ACL*, pages 117–124, Las Cruces, New Mexico.

A. Ratnaparkhi. 1997. A linear observed time statistical parser based on maximum entropy models. In *Second Conference on Empirical Methods in Natural Language Processing*, Brown University, Providence, Rhode Island.

D. J. Rosenkrantz and P. M. Lewis. 1970. Deterministic left corner parsing. In *IEEE Conf. Record 11th Annual Symposium on Switching and Automata Theory*, pages 139–152.

G. Satta and O. Stock. 1994. Bidirectional context-free grammar parsing for natural language processing. *Artificial Intelligence*, 69:123–164.

S. Sippu and E. Soisalon-Soininen. 1988. *Parsing Theory: Languages and Parsing*, volume 1. Springer-Verlag, Berlin, Germany.

S. Sippu and E. Soisalon-Soininen. 1990. *Parsing Theory: LR(k) and LL(k) Parsing*, volume 2. Springer-Verlag, Berlin, Germany.

R. E. Stearns and H. B. Hunt. 1981. On the equivalence and containment problem for unambiguous regular expressions, grammars, and automata. In *22nd Annual Symp. on Foundations of Computer Science*, pages 74–81, New York. IEEE.

M. Tomita. 1986. *Efficient Parsing for Natural Language.* Kluwer, Boston, Mass.

G. van Noord. 1997. An efficient implementation of the head-corner parser. *Computational Linguistics*, 23(3):425–456.

C. S. Wetherell. 1980. Probabilistic languages: A review and some open questions. *Computing Surveys*, 12(4):361–379.

M. Wiren. 1987. A comparison of rule-invocation strategies in parsing. In *Proc. of the 3rd EACL*, pages 226–233, Copenhagen, Denmark.

A Recognition in time independent of the lexicon

In Section 5 we have shown that it is unlikely that correct-prefix property parsing for a bilexical CFG can be carried out in polynomial time and independently of the lexicon size, when only polynomial-time off-line compilation of the grammar is allowed. To complete our presentation, we show here that correct-prefix property parsing in time independent of the lexicon size is indeed possible if we spend exponential time on grammar precompilation.

We first consider tabular LR parsing (Tomita, 1986), a technique which satisfies the correct-prefix property, and apply it to bilexical CFGs. Our presentation relies on definitions from (Nederhof and Satta, 1996). Let $w \in V_T^*$ be some input string. A property of LR parsing is that any state that can be reached after reading prefix $w[1,j]$, $j \leq |w|$, must be of the form

$$goto(goto(\ldots(goto(q_{in}, X_1), \ldots), X_{m-1}), X_m)$$

where q_{in} is the initial LR state, and $X_1, \ldots, X_m$ are terminals or nonterminals such that $X_1 \cdots X_m \Rightarrow^* w[1,j]$. For a bilexical CFG, each X_i is of the form b_i or of the form $B_i[b_i]$, where $b_1, \ldots, b_m$ is some subsequence of $w[1,j]$. This means that there are at most $(2+|V_D|)^n$ distinct states that can be reached by the recognizer, apart from q_{in}. In the algorithm, the tabulation prevents repeated manipulation of states for a triple of input positions, leading to a time complexity of $\mathcal{O}(n^3\ |V_D|^n)$, where $n = |w|$. Hence, when we apply precompilation of the grammar, we can carry out recognition in time exponential in the length of the input string, yet independent of the lexicon size. Note however that the precompilation for LR parsing takes exponential time.

The second algorithm with the CPP we will consider can be derived from Earley's algorithm (Earley, 1970). For this new recognizer, we achieve a time complexity completely independent of the size of the whole grammar, not merely independent of the size of the lexicon as in the case of tabular LR parsing. Furthermore, the input grammar can be any general CFG, not necessarily a bilexical one. In terms of the length of the input, the complexity is polynomial rather than exponential.

Earley's algorithm is outlined in what follows, with minor modifications with respect to its original presentation. An **item** is an object of the form

$[A \to \alpha \bullet \beta]$, where $A \to \alpha\beta$ is a production from the grammar. The recognition algorithm consists in an incremental construction of a $(n+1) \times (n+1)$, 2-dimensional table T, where n is the length of the input string. At each stage, each entry $T[i,j]$ in the table contains a set of items, which is initially the empty set. After an initial item is added to entry $T[0,0]$ in the table, other items in other entries are derived from it, directly or indirectly, using three steps called predictor, scanner and completer. When no more new items can be derived, the presence of a final item in entry $T[0,n]$ indicates whether the input is recognized.

The recognition process can be precompiled, based on the observation that for any grammar the set of all possible items is finite, and thereby all potential contents of T's entries can be enumerated. Furthermore, the dependence of entries on one another is not cyclic; one item in $T[i,j]$ may be derived from a second item in the same entry, but it is not possible that, for example, an item in $T[i,j]$ is derived from an item in $T[i',j']$, with $(i,j) \neq (i',j')$, which is in turn derived from an item in $T[i,j]$.

A consequence is that entries can be computed in a strict order, and an operation that involves the combination of, say, the items from two entries $T[i,j]$ and $T[j,k]$ by means of the completer step can be implemented by a simple table lookup. More precisely, each set of items is represented by an atomic state, and combining two sets of items according to the completer step is implemented by indexing a 2-dimensional array by the two states representing those two sets, yielding a third state representing the resulting set of items. Similarly, the scanner and predictor steps and the union operation on sets of items can all be implemented by table lookup.

The time complexity of recognition can straightforwardly be shown to be $\mathcal{O}(n^3)$, independent of the size of the grammar. However, massive precompilation is involved in enumerating all possible sets of items and precomputing the operations on them. The motivation for discussing this algorithm is therefore purely theoretical: it illustrates the unfavourable complexity properties that Theorem 2, together with the conjecture about quasi-determinizers, attributes to the recognition problem if the correct-prefix property is to be ensured.