Parsing Non-Recursive Context-Free Grammars

Mark-Jan Nederhof *
Faculty of Arts
University of Groningen
P.O. Box 716
NL-9700 AS Groningen, The Netherlands
markjan@let.rug.nl

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

Abstract

We consider the problem of parsing non-recursive context-free grammars, i.e., context-free grammars that generate finite languages. In natural language processing, this problem arises in several areas of application, including natural language generation, speech recognition and machine translation. We present two tabular algorithms for parsing of non-recursive context-free grammars, and show that they perform well in practical settings, despite the fact that this problem is PSPACE-complete.

1 Introduction

Several applications in natural language processing require "parsing" of a large but finite set of candidate strings. Here parsing means some computation that selects those strings out of the finite set that are well-formed according to some grammar, or that are most likely according to some language model. In these applications, the finite set is typically encoded in a compact way as a context-free grammar (CFG) that is non-recursive. This is motivated by the fact that non-recursive CFGs allow very compact representations for finite languages, since the strings derivable from single nonterminals may be substrings of many different strings in the language. Unfolding such a grammar and parsing the generated strings one by one then leads to an unnecessary duplication of subcomputations, since each occurrence of a repeated substring has to be independently parsed. As this approach may be prohibitively expensive, it is preferable to find a parsing algorithm that shares subcomputations among different strings by working directly on the nonterminals and the rules of the non-recursive CFG. In this way, "parsing" a nonterminal of the grammar amounts to shared parsing of all the substrings encoded by that nonterminal.

To give a few examples, in some natural language generation systems (Langkilde, 2000) non-recursive CFGs are used to encode very large sets of candidate sentences realizing some input conceptual representation (Langkilde calls such grammars *forests*). Each CFG is later "parsed" using a language model, in order to rank the sentences in the set according to their likelyhood. Similarly, in some approaches to automatic speech understanding (Corazza and Lavelli, 1994) the N-best sentences obtained from the speech recognition module are "compressed" into a non-recursive CFG grammar, which is later provided as input to a parser. Finally, in some machine translation applications related techniques are exploited to obtain sentences that simultaneously realize two different conceptual representations (Knight and Langkilde, 2000). This is done in order to produce translations that preserve syntactic or semantic ambiguity in cases where the ambiguity could not be resolved when processing the source sentence.

To be able to describe the above applications in an abstract way, let us first fix some terminology. The term "recognition" refers to the process of deciding

* Secondary affiliation is the German Research Center for Artificial Intelligence (DFKI).

whether an input string is in the language described by a grammar, the *parsing* grammar G_p. We will generalize this notion in a natural way to input representing a *set* of strings, and here the goal of recognition is to decide whether at least one of the strings in the set is in the language described by G_p. If the input is itself given in the form of a grammar, the *input* grammar G_i, then recognition amounts to determining whether the intersection of the languages described by G_i and G_p is non-empty. In this paper we use the term parsing as synonymous to recognition, since the recognition algorithms we present can be easily extended to yield parse trees (with associated probabilities if either G_i or G_p or both are probabilistic).

In what follows we consider the case where both G_p and G_i are CFGs. General CFGs have unfavourable computational properties with respect to intersection. In particular, the problem of deciding whether the intersection of two CFGs is non-empty is undecidable (Harrison, 1978). Following the terminology adopted above, this means that parsing a context-free input grammar G_i on the basis of a context-free parsing grammar G_p is not possible in general.

One way to make the parsing problem decidable is to place some additional restrictions on G_i or G_p. This direction is taken by Langkilde (2000), where G_i is a non-recursive CFG and G_p represents a regular language, more precisely an N-gram model. In this way the problem can be solved using a stochastic variant of an algorithm presented by Bar-Hillel et al. (1964), where it is shown that the intersection of a general context-free language and a regular language is still context-free.

In the present paper we leave the theoretical framework of Bar-Hillel et al. (1964), and consider parsing grammars G_p that are unrestricted CFGs, and input grammars G_i that are non-recursive context-free grammars. In this case the parsing (intersection) problem becomes PSPACE-complete.[1] Despite of this unfavourable theoretical result, algorithms for the problem at hand have been proposed in the literature and are currently used in practical applications. In (Knight and Langkilde, 2000) G_i is unfolded into a lattice (acyclic finite automaton) and later parsed with G_p using an algorithm close to the one by Bar-Hillel et al. (1964). The algorithm proposed by Corazza and Lavelli (1994) involves copying of charts, and this makes it very similar in behaviour to the former approach. Thus in both algorithms parts of the input grammar G_i are copied where a nonterminal occurs more than once, which destroys the compactness of the representation. In this paper we propose two alternative tabular algorithms that exploit the compactness of G_i as much as possible. Although a limited amount of copying is also done by our algorithms, this never happens in cases where the resulting structure is ungrammatical with respect to the parsing grammar G_p.

The structure of this paper is as follows. In Section 2 we introduce some preliminary definitions, followed in Section 3 by a first algorithm based on CKY parsing. A more sophisticated algorithm, satisfying the equivalent of the correct-prefix property and based on Earley's algorithm, is presented in Section 4. Section 5 presents our experimental results and Section 6 closes with some discussion.

2 Preliminaries

In this section we briefly recall some standard notions from formal language theory. For more details we refer the reader to textbooks such as (Harrison, 1978).

A context-free grammar is a 4-tuple $(\Sigma, \mathcal{N}, S, \mathcal{R})$, where Σ is a finite set of *terminals*, called the *alphabet*, $\mathcal{N}$ is a finite set of *nonterminals*, including the *start symbol* S, and $\mathcal{R}$ is a finite set of *rules* having the form $A \to \xi$ with $A \in \mathcal{N}$ and $\xi \in (\Sigma \cup \mathcal{N})^*$. Throughout the paper we assume the following conventions: $A, B, \ldots$ denote nonterminals, $a, b, \ldots$ denote terminals, μ, ν, ξ are strings in $(\Sigma \cup \mathcal{N})^*$ and v, w are strings in Σ^*. We also assume that each CFG is reduced, i.e., no CFG contains nonterminals that do not occur in any derivation of a string in the language. Furthermore, we assume that the input grammars do not contain epsilon rules and that there is only one rule $S \to \xi$ defining the start symbol S.[2] Finally, in Section 3 we will consider parsing gram-

[1] The PSPACE-hardness result has been shown by Harry B. Hunt III and Dan Rosenkrantz (Harry B. Hunt III, p.c.). Membership in PSPACE is shown by Nederhof and Satta (2002).

[2] Strictly speaking, the assumption about the absence of epsilon rules is not without loss of generality, since without epsilon rules the language cannot contain the empty string. However, this has no practical consequence.

mars in Chomsky normal form (CNF), i.e., grammars with rules of the form $A \rightarrow B\,C$ or $A \rightarrow a$.

Instead of working with non-recursive CFGs, it will be more convenient in the specification of our algorithms to encode G_i as a push-down automaton (PDA) with stack size bounded by some constant. Unlike many text-books, we assume PDAs do not have states; this is without loss of generality, since states can be encoded in the symbols that occur top-most on the stack. Thus, a PDA is a 5-tuple $(\Sigma, \mathcal{S}, X_{init}, X_{final}, \Delta)$, where Σ is the alphabet as above, $\mathcal{S}$ is a finite set of *stack symbols* including the *initial stack symbol* X_{init} and the *final stack symbol* X_{final}, and Δ is the set of *transitions*, having one of the following three forms: $X \mapsto XY$ (a push transition), $XY \mapsto Z$ (a pop transition), or $X \stackrel{a}{\mapsto} Y$ (a scan transition, scanning symbol a). Throughout this paper we use the following conventions: Q, X, Y, Z denote stack symbols and α, β, γ are strings in $\mathcal{S}^*$ representing stacks. We remark that in our notation stacks grow from left to right, i.e., the top-most stack symbol will be found at the right end.

Configurations of the PDA have the form (α, w), where $\alpha \in \mathcal{S}^*$ is a stack and $w \in \Sigma^*$ is the remaining input. We let the binary relation $\vdash$ be defined by: $(\gamma\alpha, vw) \vdash (\gamma\beta, w)$ if and only if there is a transition in Δ of the form $\alpha \mapsto \beta$, where $v = \epsilon$, or of the form $\alpha \stackrel{a}{\mapsto} \beta$, where $v = a$. The relation $\vdash^*$ denotes the reflexive and transitive closure of $\vdash$. An input string w is *recognized* by the PDA if and only if $(X_{init}, w) \vdash^* (X_{final}, \epsilon)$.

3 The CKY algorithm

In this section we present our first parsing algorithm, based on the so-called CKY algorithm (Harrison, 1978) and exploiting a decomposition of computations of PDAs cast in a specific form. We start with a construction that translates the non-recursive input CFG G_i into a PDA accepting the same language.

Let $G_i = (\Sigma, \mathcal{N}, S, \mathcal{R})$. The PDA associated with G_i is specified as

$$(\Sigma, \mathcal{S}, [S \rightarrow \bullet\, \xi], [S \rightarrow \xi\, \bullet], \Delta),$$

where $\mathcal{S}$ consists of symbols of the form $[A \rightarrow \mu \bullet \nu]$ for $(A \rightarrow \mu\nu) \in \mathcal{R}$, and Δ contains the following transitions:

- For each pair of rules $A \rightarrow \mu B \nu$ and $B \rightarrow \xi$, Δ contains:
  $[A \rightarrow \mu \bullet B\nu] \;\mapsto\; [A \rightarrow \mu \bullet B\nu]\,[B \rightarrow \bullet\, \xi]$
  and
  $[A \rightarrow \mu \bullet B\nu]\,[B \rightarrow \xi\, \bullet] \mapsto [A \rightarrow \mu B \bullet \nu]$.
- For each rule $A \rightarrow \mu a \nu$, Δ contains: $[A \rightarrow \mu \bullet a\nu] \stackrel{a}{\mapsto} [A \rightarrow \mu a \bullet \nu]$.

Observe that for all PDAs constructed as above, no push transition can be immediately followed by a pop transition, i.e., there are no stack symbols X, Y and Z such that $X \mapsto XY$ and $XY \mapsto Z$. As a consequence of this, a computation $(X_{init}, w) \vdash^* (X_{final}, \epsilon)$ of the PDA can always and uniquely be decomposed into consecutive subcomputations, which we call *segments*, each starting with zero or more push transitions, followed by a single scan transition and by zero or more pop transitions. In what follows, we will formalize this basic idea and exploit it within our parsing algorithm.

We write $\alpha \stackrel{a}{\Longrightarrow} \beta$ to indicate that there is a computation $(\alpha, a) \vdash^* (\beta, \epsilon)$ of the PDA such that all of the following three conditions hold:

(i) either $|\alpha| = 1$ or $|\beta| = 1$;

(ii) the computation starts with zero or more push transitions, followed by one scan transition reading a and by zero or more pop transitions;

(iii) if $|\alpha| > 1$ then the top-most symbol of α must be in the right-hand side of a pop or scan transition (i.e., top-most in the stack at the end of a previous segment) and if $|\beta| > 1$, then the top-most symbol of β must be the left-hand side of a push or scan transition (i.e., top-most in the stack at the beginning of a following segment).

Let $Begin = \{X_{init}\} \cup \{Z \mid \exists X, Y[XY \mapsto Z]\} \cup \{Y \mid \exists X, a[X \stackrel{a}{\mapsto} Y]\}$, and $End = \{X_{final}\} \cup \{X \mid \exists Y[X \mapsto XY]\} \cup \{X \mid \exists Y, a[X \stackrel{a}{\mapsto} Y]\}$. A formal definition of relation $\Rightarrow$ above is provided in Figure 1 by means of a deduction system. We assign a procedural interpretation to such a system following Shieber et al. (1995), resulting in an algorithm for the computation of the relation.

We now turn to an important property of segments. Any computation $(X_{init}, a_1 \cdots a_n) \vdash^* (X_{final}, \epsilon)$, $n \geq 1$, can be computed by combining

$$\frac{}{X \overset{a}{\Longrightarrow} Y} \left\{ \; X \overset{a}{\mapsto} Y \right. \tag{1}$$

$$\frac{X \overset{a}{\Longrightarrow} Y}{Q \overset{a}{\Longrightarrow} Z} \left\{ \begin{array}{l} Q \mapsto QX \\ QY \mapsto Z \end{array} \right. \tag{2}$$

$$\frac{\alpha X \overset{a}{\Longrightarrow} Y}{Q\alpha X \overset{a}{\Longrightarrow} Z} \left\{ \begin{array}{l} QY \mapsto Z \\ X \in Begin \end{array} \right. \tag{3}$$

$$\frac{X \overset{a}{\Longrightarrow} \alpha Y}{Q \overset{a}{\Longrightarrow} Q\alpha Y} \left\{ \begin{array}{l} Q \mapsto QX \\ Y \in End \end{array} \right. \tag{4}$$

Figure 1: Inference rules for the computation of relation $\Rightarrow$.

$$\frac{}{\alpha X \overset{a}{\Longrightarrow}^{+} \beta Y} \left\{ \begin{array}{l} \alpha X \overset{a}{\Longrightarrow} \beta Y \\ X \in Begin \wedge Y \in End \end{array} \right. \tag{5}$$

$$\frac{\begin{array}{l} \alpha \overset{v}{\Longrightarrow}^{+} \gamma\beta \\ \beta \overset{w}{\Longrightarrow}^{+} \delta \end{array}}{\alpha \overset{vw}{\Longrightarrow}^{+} \gamma\delta} \tag{6}$$

$$\frac{\begin{array}{l} \alpha \overset{v}{\Longrightarrow}^{+} \beta \\ \gamma\beta \overset{w}{\Longrightarrow}^{+} \delta \end{array}}{\gamma\alpha \overset{vw}{\Longrightarrow}^{+} \delta} \tag{7}$$

Figure 2: Inference rules for combining segments $\alpha_i \overset{a_i}{\Longrightarrow} \beta_i$.

n segments represented by $\alpha_i \overset{a_i}{\Longrightarrow} \beta_i$, $1 \leq i \leq n$, with $\alpha_1 = X_{init}$, $\beta_n = X_{final}$, and for $1 \leq i < n$, β_i is a suffix of α_{i+1} or α_{i+1} is a suffix of β_i. This is done by the deduction system given in Figure 2, which defines the relation $\Longrightarrow^{+}$. The second side-condition of inference rule (5) checks whether a segment $\alpha X \overset{a}{\Longrightarrow} \beta Y$ may border on other segments, or may be the first or last segment in a computation.

Figure 3 illustrates a computation of a PDA recognizing a string $a_1a_2a_3a_4$. A horizontal line segment in the curve represents a scan transition, an upward line segment represents a push transition, and a downward line segment a pop transition. The shaded areas represent segments $\alpha_i \overset{a_i}{\Longrightarrow} \beta_i$. As an example, the area labelled I represents $X_{init} \overset{a_1}{\Longrightarrow} X_{init}X_1X_2$, for certain stack symbols X_1 and X_2, where the left edge of the shaded area represents X_{init} and the right edge represents $X_{init}X_1X_2$. Note that segments $\alpha_i \overset{a_i}{\Longrightarrow} \beta_i$ abstract away from the stack symbols that are pushed and then popped again. Furthermore, in the context of the whole computation, segments abstract away from stack symbols that are not accessed during a subcomputation. As an example, the shaded area labelled III represents segment $Y_1Y_2 \overset{a_3}{\Longrightarrow} Z$, for certain stack symbols Y_1, Y_2 and Z, and this abstracts away from the stack symbols that may occur below Y_1 and Z.

Figure 4 illustrates how two adjacent segments are combined. The dashed box in the left-hand side of the picture represents stack symbols from the right edge of segment II that need not be explicitly represented by segment III, as discussed above. We may assume that these symbols exist, so that II and III can be combined into the larger computation in the right-hand side of the picture. Note that if a computation $\alpha \overset{w}{\Longrightarrow}^{+} \beta$ is obtained as the combination of two segments as in Figure 4, then some internal details of these segments are abstracted away, i.e., stack elements that were pushed and again popped in the combined computation are no longer recorded. This abstraction is a key feature of the parsing algorithm to be presented next, in that it considerably reduces the time complexity as compared with that of an algorithm that investigates all computations of the PDA in isolation.

We are now ready to present our parsing algorithm, which is the main result of this section. The algorithm combines the deduction system in Figure 2, as applied to the PDA encoding the input grammar G_i, with the CKY algorithm as applied to the parsing grammar G_p. (We assume that G_p is in CNF.) The parsing algorithm may rule out many combinations of segments from Figure 2 that are inconsistent with the language generated by G_p. Also ruled out are structural compositions of segments that are inconsistent with the structure that G_p assigns to the corresponding substrings.

The parsing algorithm is again specified as a deduction system, presented in Figure 5. The algorithm manipulates items of the form $[A, \alpha, \beta]$, where A is a nonterminal of G_p and α, β are stacks of the PDA encoding G_i. Such an item indicates that there

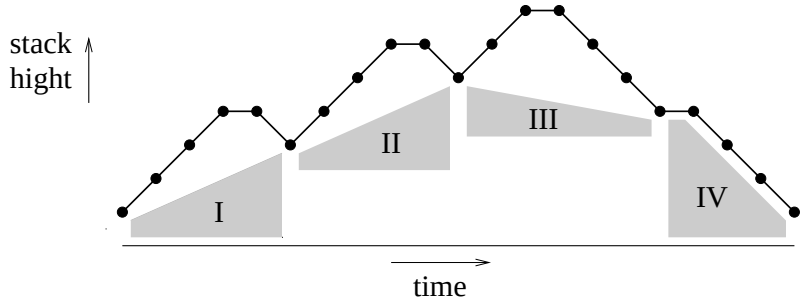

Figure 3: A computation of a PDA divided into segments.

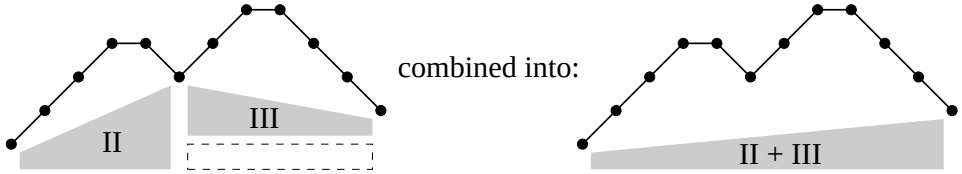

Figure 4: Combining two segments using rule (6) from Figure 2.

$$\frac{\left.\begin{array}{l}\alpha X \stackrel{a}{\Longrightarrow} \beta Y \\ X \in Begin \wedge Y \in End \\ A \rightarrow a\end{array}\right\}}{[A, \alpha X, \beta Y]} \quad (8)$$

$$\frac{[B, \alpha, \gamma\beta] \quad [C, \beta, \delta]}{[A, \alpha, \gamma\delta]} \left\{ A \rightarrow B\,C \right. \quad (9)$$

$$\frac{[B, \alpha, \beta] \quad [C, \gamma\beta, \delta]}{[A, \gamma\alpha, \delta]} \left\{ A \rightarrow B\,C \right. \quad (10)$$

Figure 5: Inference rules that simultaneously derive strings generated by G_p and accepted by the PDA encoding G_i.

is some terminal string w that is derivable from A in G_p, and such that $(\alpha, w) \vdash^* (\beta, \epsilon)$. If the item $[S, X_{init}, X_{final}]$ can be derived by the algorithm, then the intersection of the language generated by G_p and the language accepted by the PDA (generated by G_i) is non-empty.

4 Earley's algorithm

The CKY algorithm from Figure 5 can be seen to filter out a selection of the computations that may be derived by the deduction system from Figure 2. One may however be even more selective in determining which computations of the PDA to consider. The basis for the algorithm in this section is Earley's algorithm (Earley, 1970). This algorithm differs from the CKY algorithm in that it satisfies the *correct-prefix property* (Harrison, 1978).

The new algorithm is presented by Figure 6. There are now two types of item involved. The first item has the form $[A \rightarrow \mu \bullet \nu \mid \gamma * \alpha, \gamma * \beta]$, where $A \rightarrow \mu \bullet \nu$ has the same role as the dotted rules in Earley's original algorithm. The second and third components are stacks of the PDA as before, but these stacks now contain a distinguished position, indicated by $*$. The existence of an item $[A \rightarrow \mu \bullet \nu \mid \gamma * \alpha, \gamma * \beta]$ implies that $(\gamma\alpha, v) \vdash^* (\gamma\beta, \epsilon)$, where v is now a string derivable from μ. This is quite similar to the meaning we assigned to the items of the CKY algorithm, but here not all stack symbols in $\gamma\alpha$ and $\gamma\beta$ are involved in this computation: only the symbols in α and β are now accessed, while all symbols in γ remain unaffected. The portion of the stack represented by γ is needed to ensure the correct-prefix property in subsequent computations following from this item, in case all of the symbols in β are popped.

The correct-prefix property is ensured in the following sense. The existence of an item $[A \rightarrow \mu \bullet \nu \mid \gamma * \alpha, \gamma * \beta]$ implies that (i) there is a string wv that is both a prefix of a string accepted by the PDA and of a string generated by the CFG such that after

$$\frac{}{[S \to \bullet\, \xi \mid *\, X_{init}, *\, X_{init}]} \left\{ S \to \xi \right. \quad (11)$$

$$\frac{[A \to \mu \bullet a\nu \mid *\, \alpha, *\, \gamma\beta]}{[A \to \mu a \bullet \nu \mid *\, \alpha, *\, \gamma\delta]} \left\{ \beta \overset{a}{\Longrightarrow}^{+} \delta \right. \quad (12)$$

$$\frac{[A \to \mu \bullet a\nu \mid \gamma * \alpha, \gamma * \beta]}{[A \to \mu a \bullet \nu \mid *\, \gamma\alpha, *\, \delta]} \left\{ \gamma\beta \overset{a}{\Longrightarrow}^{+} \delta \right. \quad (13)$$

$$\frac{[A \to \mu \bullet a\nu \mid *\, \alpha, *\, \beta]}{[A \to \mu \bullet a\nu \mid *\, \alpha, *\, \beta \mid Q?]} \left\{ \gamma Q\beta \overset{a}{\Longrightarrow}^{+} \delta \right. \quad (14)$$

$$\frac{[A \to \mu \bullet B\nu \mid *\, \alpha, *\, \beta X]}{[B \to \bullet\, \xi \mid *\, X, *\, X]} \left\{ B \to \xi \right. \quad (15)$$

$$\frac{\begin{array}{c}[A \to \mu \bullet B\nu \mid *\, \alpha, *\, \gamma\beta] \\ [B \to \xi \bullet \mid *\, \beta, *\, \delta]\end{array}}{[A \to \mu B \bullet \nu \mid *\, \alpha, *\, \gamma\delta]} \quad (16)$$

$$\frac{\begin{array}{c}[A \to \mu \bullet B\nu \mid \gamma * \alpha, \gamma * \beta] \\ [B \to \xi \bullet \mid *\, \gamma\beta, *\, \delta]\end{array}}{[A \to \mu B \bullet \nu \mid *\, \gamma\alpha, *\, \delta]} \quad (17)$$

$$\frac{[A \to \mu \bullet \nu \mid \alpha X, \beta \mid Q?]}{[A \to \bullet\, \mu\nu \mid \alpha * X, \alpha * X \mid Q?]} \quad (18)$$

$$\frac{\begin{array}{c}[A \to \mu \bullet B\nu \mid \alpha, \beta X] \\ [B \to \bullet\, \xi \mid \beta * X, \beta * X \mid Q?]\end{array}}{[A \to \mu \bullet B\nu \mid \alpha, \beta X \mid Q?]} \quad (19)$$

$$\frac{\begin{array}{c}[A \to \mu \bullet B\nu \mid \alpha, \gamma Q\beta X] \\ [B \to \bullet\, \xi \mid \beta * X, \beta * X \mid Q?]\end{array}}{[B \to \bullet\, \xi \mid Q\beta * X, Q\beta * X]} \quad (20)$$

$$\frac{\begin{array}{c}[A \to \bullet\, \mu\nu \mid Q\alpha_1\alpha_2 * X, Q\alpha_1\alpha_2 * X] \\ [A \to \mu \bullet \nu \mid \alpha_1 * \alpha_2 X, \alpha_1 * \beta \mid Q?]\end{array}}{[A \to \mu \bullet \nu \mid Q\alpha_1 * \alpha_2 X, Q\alpha_1 * \beta]} \quad (21)$$

Figure 6: Inference rules based on Earley's algorithm.

processing w, A is expanded in a left-most derivation and some stack can be obtained of which $\gamma\alpha$ represent the top-most elements, and (ii) μ is rewritten to v and while processing v the PDA replaces the stack elements α by β.[3]

The second type of item has the form $[A \to \mu \bullet \nu \mid \gamma * \alpha, \gamma * \beta \mid Q?]$. The first three components are the same as before, and Q indicates that we wish to know whether a stack with top-most symbols $Q\gamma\alpha$ may arise after reading a prefix of a string that may also lead to expansion of nonterminal A in a left-most derivation. Such an item results if it is detected that the existence of Q below $\gamma\alpha$ needs to be ensured in order to continue the computation under the constraint of the correct-prefix property.

Our algorithm also makes use of segments, as computed by the algorithm from Figure 1. Consistently with rule (5) from Figure 2, we write $\alpha X \overset{a}{\Longrightarrow}^{+} \beta Y$ to represent a segment $\alpha X \overset{a}{\Longrightarrow} \beta Y$ such that $X \in \mathit{Begin} \wedge Y \in \mathit{End}$. The use of segments that were computed bottom-up is a departure from pure left-to-right processing in the spirit of Earley's original algorithm. The motivation is that we have found empirically that the use of rule (2) was essential for avoiding a large part of the exponential behaviour; note that that rule considers at most a number of stacks that is quadratic in the size of the PDA.

The first inference rule (11) can be easily justified: we want to investigate strings that are both generated by the grammar and recognized by the PDA, so we begin by combining the start symbol and a matching right-hand side from the grammar with the initial stack for the PDA.

Segments are incorporated into the left-to-right computation by rules (12) and (13). These two rules are the equivalents of (9) and (10) from Figure 5. Note that in the case of (13) we require the presence of γ below the marker in the antecedent. This indicates that a stack with top-most symbols $\gamma\alpha$ and a dotted rule $A \to \mu \bullet a\nu$ can be obtained by simultaneously processing a string from left to right by the grammar and the PDA. Thereby, we may continue the derivation with the item in the consequent without violating the correct-prefix property.

Rule (14) states that if a segment presupposes the existence of stack elements that are not yet available, we produce an item that starts a backward computation. We do this one symbol at a time, starting with

[3] We naturally assume that the PDA itself satisfies the correct-prefix property, which is guaranteed by the construction from Section 3 and the fact that G_i is reduced.

the symbol Q just beneath the part of the stack that is already available. This will be discussed more carefully below.

The predictor step of Earley's algorithm is represented by (15), and the completer step by rules (16) and (17). These latter two are very similar to (12) and (13) in that they incorporate a smaller derivation in a larger derivation.

Rules (18) and (19) repeat computations that have been done before, but in a backward manner, in order to propagate the information that deeper stack symbols are needed than those currently available, in particular that we want to know whether a certain stack symbol Q may occur below the currently available parts of the stack. In (18) this query is passed on to the beginning of the context-free rule, and in (19) this query is passed on backwards through a predictor step. In the antecedent of rule (18) the position of the marker is irrelevant, and is not indicated explicitly. Similarly, for rule (19) we assume the position of the marker is copied unaltered from the first antecedent to the consequent.

If we find the required stack symbol Q, we propagate the information forward that this symbol may indeed occur at the specified position in the stack. This is implemented by rules (20) and (21). Rule (20) corresponds to the predictor step (15), but (20) passes on a larger portion of the stack than (20). Rule (15) only transfers the top-most symbol X to the consequent, in order to keep the stacks as shallow as possible and to achieve a high degree of sharing of computation.

5 Empirical results

We have implemented the two algorithms and tested them on non-recursive input CFGs and a parsing CFG. We have had access to six input CFGs of the form described by Langkilde (2000). As parsing CFG we have taken a small hand-written grammar of about 100 rules. While this small size is not at all typical of practical grammars, it suffices to demonstrate the applicability of our algorithms.

The results of the experiments are reported in Figure 1. We have ordered the input grammars by size, according to the number of nonterminals (or the number of nodes in the forest, following the terminology by Langkilde (2000)).

The second column presents the number of strings generated by the input CFG, or more accurately, the number of derivations, as the grammars contain some ambiguity. The high numbers show that without a doubt the naive solution of processing the input grammars by enumerating individual strings (derivations) is not a viable option.

The third column shows the size, expressed as number of states, of a lattice (acyclic finite automaton) that would result by unfolding the grammar (Knight and Langkilde, 2000). Although this approach could be of more practical interest than the naive approach of enumerating all strings, it still leads to large intermediate results. In fact, practical context-free parsing algorithms for finite automata have cubic time complexity in the number of states, and derive a number of items that is quadratic in the number of states.

The next column presents the number of segments $\alpha \stackrel{a}{\Longrightarrow} \beta$. These apply to both algorithm. We only compute segments $\alpha \stackrel{a}{\Longrightarrow} \beta$ for terminals a that also occur in the parsing grammar. (Further obvious optimizations in the case of Earley's algorithm were found to lead to no more than a slight reduction of produced segments.) The last two columns present the number of items specific to the two algorithms in Figures 5 and 6, respectively. Although our two algorithms are exponential in the number of stack symbols in the worst case, just as approaches that enumerate all strings or that unfold G_i into a lattice, we see that the numbers of items are relatively moderate if we compare them to the number of strings generated by the input grammars.

Earley's algorithm generally produces more items than the CKY algorithm. An exception is the last input CFG; it seems that the number of items that Earley's algorithm needs to consider in order to maintain the correct-prefix property is very sensitive to qualities of the particular input CFG.

The present implementations use a trie to store stacks; the arcs in the trie closest to the root represent stack symbols closest to the top of the stacks. For example, for storing $\alpha \stackrel{a}{\Longrightarrow} \beta$, the algorithm represents α and β by their corresponding nodes in the trie, and it indexes $\alpha \stackrel{a}{\Longrightarrow} \beta$ twice, once through each associated node. Since the trie is doubly linked (i.e. we may traverse the trie upwards as well as downwards), we can always reconstruct the stacks

Table 1: Empirical results.

# nonts	# strings	# states	# segments	# items CKY	# items Earley
168	$3.9 * 10^7$	2643	1437	1252	6969
248	$9.9 * 10^8$	21984	3542	4430	40568
259	$3.0 * 10^9$	6528	957	1314	29925
361	$1.7 * 10^{11}$	77198	7824	14627	14907
586	$3.9 * 10^{12}$	45713	8832	5608	8611
869	$7.7 * 10^{12}$	63851	15679	5709	3781

from the corresponding nodes. This structure is also convenient for finding pairs of matching stacks, one of which may be deeper than the other, as required by the inference rules from e.g. Figure 5, since given the first stack in such a pair, the second can be found by traversing the trie either upwards or downwards.

6 Discussion

It is straightforward to give an algorithm for parsing a finite language: we may trivially parse each string in the language in isolation. However, this is not a practical solution when the number of strings in the language exceeds all reasonable bounds.

Some algorithms have been described in the existing literature that parse sets of strings of exponential size in the length of the input description. These solutions have not considered context-free parsing of finite languages encoded by non-recursive CFGs, in a way that takes full advantage of the compactness of the representation. Our algorithms make this possible, relying on the compactness of the input grammars for efficiency in practical cases, and on the absence of recursion for guaranteeing termination. Our experiments also show that these algorithms are of practical interest.

Acknowledgements

We are indebted to Irene Langkilde for putting to our disposal the non-recursive CFGs on which we have based our empirical evaluation.

References

Y. Bar-Hillel, M. Perles, and E. Shamir. 1964. 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.

A. Corazza and A. Lavelli. 1994. An N-best representation for bidirectional parsing strategies. In *Working Notes of the AAAI'94 Workshop on Integration of Natural Language and Speech Processing*, pages 7–14, Seattle, WA.

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

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

K. Knight and I. Langkilde. 2000. Preserving ambiguities in generation via automata intersection. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 697–702, Austin, Texas, USA, July–August.

I. Langkilde. 2000. Forest-based statistical sentence generation. In *6th Applied Natural Language Processing Conference and 1st Meeting of the North American Chapter of the Association for Computational Linguistics*, pages Section 2, 170–177, Seattle, Washington, USA, April–May.

M.-J. Nederhof and G. Satta. 2002. The emptiness problem for intersection of a CFG and a nonrecursive CFG is PSPACE-complete. In preparation.

S.M. Shieber, Y. Schabes, and F.C.N. Pereira. 1995. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24:3–36.