Context-Free Parsing through Regular Approximation

Mark-Jan Nederhof

DFKI
Stuhlsatzenhausweg 3, D-66123 Saarbrücken, GERMANY

Abstract. We show that context-free parsing can be realised by a 2-phase process, relying on an approximated context-free grammar. In the first phase a finite transducer performs parsing according to the approximation. In the second phase, the approximated parses are refined according to the original grammar.

1 Introduction

A recent publication [15] presented a novel way of transforming a context-free grammar into a new grammar that generates a regular language. This new language is a superset of the original language. It was argued that this approach has advantages over other methods of regular approximation [16, 7].

Our method of approximation is the following. We define a condition on context-free grammars that is a sufficient condition for a grammar to generate a regular language. We then give a transformation that turns an arbitrary grammar into another grammar that satisfies this condition. This transformation is obviously not language-preserving; it adds strings to the language generated by the original grammar, in such a way that the language becomes regular.

In the present communication we show how this procedure needs to be extended so that context-free parsing can be realised by a 2-phase process. For the first phase, the approximated grammar is turned into a finite transducer. This transducer processes the input in linear time and produces a table. In the second phase, this table is processed to obtain the set of all parses according to the original grammar.

The order of the time complexity of the second phase is cubic, which corresponds to the time complexity of most context-free parsing algorithms that are used in practice. However, the first phase filters out many parses that are inconsistent with respect to the regular approximation. This may reduce the effort needed by the second phase.

It is interesting to note that the work presented here is conceptually related to use of regular lookahead in context-free parsing [5].

The structure of this paper is as follows. In Section 2 we recall some standard definitions from language theory. Section 3 investigates a sufficient condition for a context-free grammar to generate a regular language. We also present the construction of a finite transducer from such a grammar.

How this transducer reads input and how the output of the transducer can be turned into a representation of all parse trees is discussed in Sections 4 and 5, respectively.

An algorithm to transform a grammar if the sufficient condition mentioned above is not satisfied is given in Section 6. Section 7 explains how this transformation can be incorporated into the construction of the transducer and how the output of such a transducer is then to be interpreted in order to obtain parse trees according to the original grammar.

Some preliminary conclusions drawn from empirical results are given in Section 8.

2 Preliminaries

A *context-free grammar* G is a 4-tuple (Σ, N, P, S), where Σ and N are two finite disjoint sets of terminals and nonterminals, respectively, $S \in N$ is the start symbol, and P is a finite set of rules. Each rule has the form $A \to \alpha$ with $A \in N$ and $\alpha \in V^*$, where V denotes $N \cup \Sigma$. The relation $\to$ on $N \times V^*$ is extended to a relation on $V^* \times V^*$ as usual. The transitive and reflexive closure of $\to$ is denoted by $\to^*$.

The language *generated* by a context-free grammar is given by the set $\{w \in \Sigma^* \mid S \to^* w\}$. By definition, such a set is a *context-free* language. By *reduction* of a grammar we mean the elimination from P of all rules $A \to \gamma$ such that $S \to^* \alpha A \beta \to \alpha\gamma\beta \to^* w$ does not hold for any $\alpha, \beta \in V^*$ and $w \in \Sigma^*$.

We generally use symbols $A, B, C, \ldots$ to range over N, symbols $a, b, c, \ldots$ to range over Σ, symbols X, Y, Z to range over V, symbols $\alpha, \beta, \gamma, \ldots$ to range over V^*, and symbols $v, w, x, \ldots$ to range over Σ^*. We write ε to denote the empty string.

A rule of the form $A \to B$ is called a *unit* rule.

A (nondeterministic) *finite automaton* $\mathcal{F}$ is a 5-tuple $(K, \Sigma, \Delta, s, F)$, where K is a finite set of *states*, of which s is the *initial state* and those in $F \subseteq K$ are the *final states*, Σ is the input alphabet, and the *transition relation* Δ is a finite subset of $K \times \Sigma^* \times K$.

We define a *configuration* to be an element of $K \times \Sigma^*$. We define the binary relation $\vdash$ between configurations as: $(q, vw) \vdash (q', w)$ if and only if $(q, v, q') \in \Delta$. The transitive and reflexive closure of $\vdash$ is denoted by $\vdash^*$.

Some input v is *recognized* if $(s, v) \vdash^* (q, \varepsilon)$, for some $q \in F$. The language *accepted* by $\mathcal{F}$ is defined to be the set of all strings v that are recognized. By definition, a language accepted by a finite automaton is called a *regular* language.

A *finite transducer* $\mathcal{T}$ is a 6-tuple $(K, \Sigma_1, \Sigma_2, \Delta, s, F)$. Next to the input alphabet Σ_1 we now have an output alphabet Σ_2. Transitions are of the form $(q, v|w, q')$ where $v \in \Sigma_1^*$ and $w \in \Sigma_2^*$.

For finite transducers, a configuration is an element of $K \times \Sigma_1^* \times \Sigma_2^*$. We define the binary relation $\vdash$ between configurations as: $(q, v_1 w_1, w_2) \vdash (q', w_1, w_2 v_2)$ if and only if $(q, v_1|v_2, q') \in \Delta$.

Some input w_1 is associated with output w_2 if $(s, w_1, \varepsilon) \vdash^* (q, \varepsilon, w_2)$, for some $q \in F$. The set of all such pairs (w_1, w_2) is the *(regular) transduction* represented by the transducer.

3 The Structure of Parse Trees

We define a *spine* in a parse tree to be a path that runs from the root down to some leaf. Our main interest in spines lies in the sequences of grammar symbols at nodes bordering on spines.

A simple example is the set of parse trees such as the one in Figure 1 (a), for a 3-line grammar of palindromes. It is intuitively clear that the language is not regular: the grammar symbols to the left of the spine from the root to ε "communicate" with those to the right of the spine. More precisely, the prefix of the input up to the point where it meets the final node ε of the spine determines the suffix after that point, in a way that an unbounded quantity of symbols from the prefix need to be taken into account.

A formal explanation for why the grammar may not generate a regular language relies on the following definition [4]:

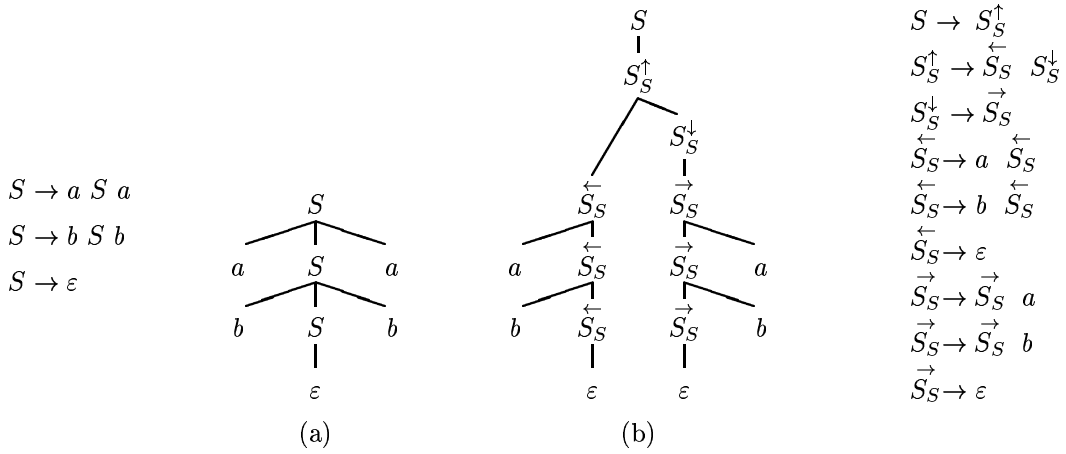

Figure 1. Parse trees for a palindrome: (a) original grammar, (b) transformed grammar (Section 6).

Definition 1 *A grammar is* self-embedding *if there is some* $A \in N$ *such that* $A \to^* \alpha A \beta$, *for some* $\alpha \neq \varepsilon$ *and* $\beta \neq \varepsilon$.

In order to avoid the somewhat unfortunate term *nonself-embedding* (or *noncenter-embedding* [11]) we define a *strongly regular* grammar to be a grammar that is not self-embedding. Strong regularity informally means that when a section of a spine in a parse tree repeats itself, then either no grammar symbols occur to the left of that section of the spine, or no grammar symbols occur to the right. This prevents the "unbounded communication" between the two sides of the spine exemplified by the palindrome grammar.

We now prove that strongly regular grammars generate regular languages. For an arbitrary grammar, we define the set of *recursive* nonterminals as:

$$\overline{N} = \{A \in N \mid \exists \alpha, \beta [A \to^* \alpha A \beta]\}$$

We determine the partition $\mathcal{N}$ of $\overline{N}$ consisting of subsets $N_1, N_2, \ldots, N_k$, for some $k \geq 0$, of *mutually recursive* nonterminals:

$$\mathcal{N} = \{N_1, N_2, \ldots, N_k\}$$
$$N_1 \cup N_2 \cup \ldots \cup N_k = \overline{N}$$
$$\forall i [N_i \neq \emptyset] \text{ and } \forall i, j [i \neq j \Rightarrow N_i \cap N_j = \emptyset]$$
$$\exists i [A \in N_i \wedge B \in N_i] \quad \Leftrightarrow \quad \exists \alpha_1, \beta_1, \alpha_2, \beta_2 [A \to^* \alpha_1 B \beta_1 \wedge B \to^* \alpha_2 A \beta_2], \text{ for all } A, B \in \overline{N}$$

We now define the function *recursive* from $\mathcal{N}$ to the set $\{left, right, self, cyclic\}$:

$$\begin{aligned} recursive(N_i) &= left, && if\ \neg LeftGenerating(N_i) \wedge\ \ RightGenerating(N_i) \\ &= right, && if\ \ LeftGenerating(N_i) \wedge \neg RightGenerating(N_i) \\ &= self, && if\ \ LeftGenerating(N_i) \wedge\ \ RightGenerating(N_i) \\ &= cyclic, && if\ \neg LeftGenerating(N_i) \wedge \neg RightGenerating(N_i) \end{aligned}$$

where

$$\mathit{LeftGenerating}(N_i) = \exists(A \to \alpha B \beta) \in P[A \in N_i \wedge B \in N_i \wedge \alpha \neq \varepsilon]$$
$$\mathit{RightGenerating}(N_i) = \exists(A \to \alpha B \beta) \in P[A \in N_i \wedge B \in N_i \wedge \beta \neq \varepsilon]$$

When $\mathit{recursive}(N_i) = \mathit{left}$, N_i consists of only left-recursive nonterminals, which does not mean it cannot also contain right-recursive nonterminals, but in that case right recursion amounts to application of unit rules. When $\mathit{recursive}(N_i) = \mathit{cyclic}$, it is *only* such unit rules that take part in the recursion.

That $\mathit{recursive}(N_i) = \mathit{self}$, for some i, is a sufficient and necessary condition for the grammar to be self-embedding. Therefore, we have to prove that if $\mathit{recursive}(N_i) \in \{\mathit{left}, \mathit{right}, \mathit{cyclic}\}$, for all i, then the grammar generates a regular language. Our proof differs from an existing proof [3] in that it is fully constructive: Figure 2 presents an algorithm for creating a finite transducer that recognizes as input all strings from the language generated by the grammar, and produces output strings of a form to be discussed shortly.

The process is initiated at the start symbol, and from there the process descends the grammar in all ways until terminals are encountered. Descending the grammar is straightforward in the case of rules of which the left-hand side is not a recursive nonterminal: the subautomata found recursively for members in the right-hand side will be connected. In the case of recursive nonterminals, the process depends on whether the nonterminals in the corresponding set from $\mathcal{N}$ are mutually left-recursive or right-recursive; if they are both, which means they are cyclic, then either subprocess can be applied; in the code in Figure 2 cyclic and left-recursive subsets N_i are treated uniformly.

We discuss the case that the nonterminals are left-recursive or cyclic. One new state is created for each nonterminal in the set. The transitions that are created for terminals and nonterminals not in N_i are connected in a way that is reminiscent of the construction of left-corner parsers [17].

The output of the transducer consists of a list of *filter items* interspersed with input symbols. A filter item is a rule with a distinguished position in the right-hand side, indicated by a diamond. The part to the left of the diamond generates a part of the input just to the left of the current input position. The part to the right of the diamond potentially generates a subsequent part of the input. A string consisting of filter items and input symbols can be seen as a representation of a parse, different from some existing representations [11, 9, 12].

At this point we use only *initial* filter items, from the set:

$$\begin{aligned} I_{init} = \{&B \to X_1 \ldots X_{m-1} \diamond X_m \mid (B \to X_1 \ldots X_{m-1} X_m) \in P \wedge \\ &\exists i[\mathit{recursive}(N_i) = \mathit{right} \wedge B, X_m \in N_i \wedge X_1, \ldots, X_{m-1} \notin N_i]\} \cup \\ \{&B \to X_1 \ldots X_m \diamond \mid (B \to X_1 \ldots X_m) \in P \wedge \\ &(m = 0 \vee \neg \exists i[\mathit{recursive}(N_i) = \mathit{right} \wedge B, X_m \in N_i \wedge X_1, \ldots, X_{m-1} \notin N_i])\} \end{aligned}$$

This definition implies that for every rule there is exactly one initial filter item. The diamond holds the rightmost position, unless we are dealing with a right-recursive rule.

An example is given in Figure 3. Four states have been labelled according to the names they are given in procedure *make_fst*. There are two states that are labelled q_B. This can be explained by the fact that nonterminal B can be reached by descending the grammar from S in two essentially distinct ways.

let $K = \emptyset$, $\Delta = \emptyset$, $s = \mathit{fresh\_state}$, $f = \mathit{fresh\_state}$, $F = \{f\}$; $\mathit{make\_fst}(s, S, f)$.

procedure $\mathit{make\_fst}(q_0, \alpha, q_1)$:
if $\alpha = \varepsilon$
then let $\Delta = \Delta \cup \{(q_0, \varepsilon|\varepsilon, q_1)\}$
elseif $\alpha = a$, **some** $a \in \Sigma$
then let $\Delta = \Delta \cup \{(q_0, a|a, q_1)\}$
elseif $\alpha = X\beta$, **some** $X \in V$, $\beta \in V^*$ **such that** $|\beta| > 0$
then let $q = \mathit{fresh\_state}$; $\mathit{make\_fst}(q_0, X, q)$; $\mathit{make\_fst}(q, \beta, q_1)$
else let $A = \alpha$; (* α must consist of a single nonterminal *)
if $A \in N_i$, **some** i
then for each $B \in N_i$ **do let** $q_B = \mathit{fresh\_state}$ **end**;
if $\mathit{recursive}(N_i) = \mathit{right}$
then for each $(B \to X_1 \ldots X_m) \in P$ **such that** $B \in N_i \wedge X_1, \ldots, X_m \notin N_i$
do let $q = \mathit{fresh\_state}$; $\mathit{make\_fst}(q_B, X_1 \ldots X_m, q)$;
let $\Delta = \Delta \cup \{(q, \varepsilon|(B \to X_1 \ldots X_m \diamond), q_1)\}$
end;
for each $(B \to X_1 \ldots X_m C) \in P$ **such that** $B, C \in N_i \wedge X_1, \ldots, X_m \notin N_i$
do let $q = \mathit{fresh\_state}$; $\mathit{make\_fst}(q_B, X_1 \ldots X_m, q)$;
let $\Delta = \Delta \cup \{(q, \varepsilon|(B \to X_1 \ldots X_m \diamond C), q_C)\}$
end;
let $\Delta = \Delta \cup \{(q_0, \varepsilon|\varepsilon, q_A)\}$
else for each $(B \to X_1 \ldots X_m) \in P$ **such that** $B \in N_i \wedge X_1, \ldots, X_m \notin N_i$
do let $q = \mathit{fresh\_state}$; $\mathit{make\_fst}(q_0, X_1 \ldots X_m, q)$;
let $\Delta = \Delta \cup \{(q, \varepsilon|(B \to X_1 \ldots X_m \diamond), q_B)\}$
end;
for each $(B \to C X_1 \ldots X_m) \in P$ **such that** $B, C \in N_i \wedge X_1, \ldots, X_m \notin N_i$
do let $q = \mathit{fresh\_state}$; $\mathit{make\_fst}(q_C, X_1 \ldots X_m, q)$;
let $\Delta = \Delta \cup \{(q, \varepsilon|(B \to C X_1 \ldots X_m \diamond), q_B)\}$
end;
let $\Delta = \Delta \cup \{(q_A, \varepsilon|\varepsilon, q_1)\}$
end
else for each $(A \to \beta) \in P$ (* A is not recursive *)
do let $q = \mathit{fresh\_state}$; $\mathit{make\_fst}(q_0, \beta, q)$; **let** $\Delta = \Delta \cup \{(q, \varepsilon|(A \to \beta \diamond), q_1)\}$
end
end
end.

procedure $\mathit{fresh\_state}()$:
create some fresh object q; **let** $K = K \cup \{q\}$; **return** q
end.

Figure 2. Transformation from a strongly regular grammar $G = (\Sigma, N, P, S)$ to a finite transducer $\mathcal{T} = (K, \Sigma, \Sigma \cup I_{init}, \Delta, s, F)$.

4 Tabular Simulation of Finite Transducers

After a finite transducer has been obtained, it may sometimes be turned into a deterministic transducer [13]. However, this is not always possible since not all regular transductions can be

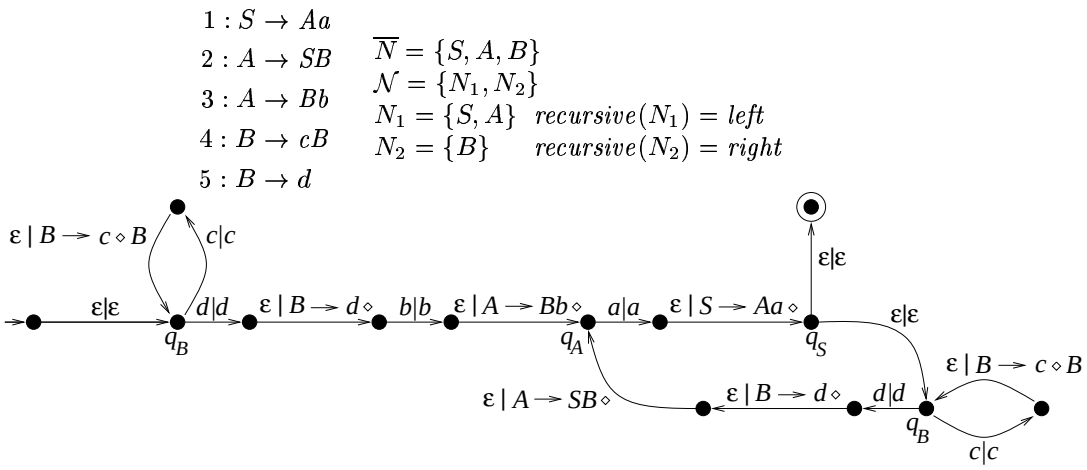

Figure 3. Application of the code from Figure 2 on a small grammar.

described by means of deterministic finite transducers. In this case, input can be processed by simulating a nondeterministic transducer in a tabular way.

Assume we have a finite transducer $\mathcal{T} = (K, \Sigma_1, \Sigma_2, \Delta, s, F)$ and an input string $a_1 \cdots a_n$. We create two tables. The first table K' contains entries of the form (i, q_1), where $0 \leq i \leq n$ and $q_1 \in K$. Such an entry indicates that the transducer may be in state q_1 after reading input from position 0 up to i. The second table Δ' contains entries of the form $((i, q_1), v, (j, q_2))$, where $v \in \Sigma_2^*$. Such an entry indicates that furthermore the transducer may go from state q_1 to q_2 in a single step, by reading the input from position i to position j while producing v as output.

The preferred way of looking at these two tables is as a set of states and a set of transitions of a finite automaton $\mathcal{F} = (K', \Sigma_2, \Delta', (0, s), F')$, where F' is a subset of $\{n\} \times F$.

Initially $K' = \{(0, s)\}$ and $\Delta' = \emptyset$. Then the following is repeated until no more new elements can be added to K' or Δ':

1. We choose a state $(i, q_1) \in K'$ and a transition $(q_1, a_{i+1} \cdots a_j | v, q_2) \in \Delta$.
2. We add a state (j, q_2) to K' and a transition $((i, q_1), v, (j, q_2))$ to Δ' if not already present.

We then define $F' = K' \cap (\{n\} \times F)$. The input $a_1 \cdots a_n$ is recognized by $\mathcal{T}$ when F' is non-empty. The language accepted by $\mathcal{F}$ is the set of output strings that $a_1 \cdots a_n$ is associated with by $\mathcal{T}$ [1].

Before continuing with the next phases of processing, as presented in the following sections, we may first reduce the automaton, i.e. we may remove the transitions that do not contribute to any paths from $(0, s)$ to a state in F'. For simplifying the discussion in the next section, we further assume that $\mathcal{F}$ is transformed such that all transitions $(q, v, q') \in \Delta'$ satisfy $|v| = 1$.

For the running example of Figure 3 we may then obtain the finite automaton indicated by the thick lines in Figure 4. (Two ε-transitions were implicitly eliminated and the automaton has been reduced.)

The time demand of the construction of $\mathcal{F}$ from $\mathcal{T}$ and $a_1 \cdots a_n$ is linear measured both in n and in the size of $\mathcal{T}$. Note that in general the language accepted by $\mathcal{F}$ may be infinite in case the grammar is cyclic.

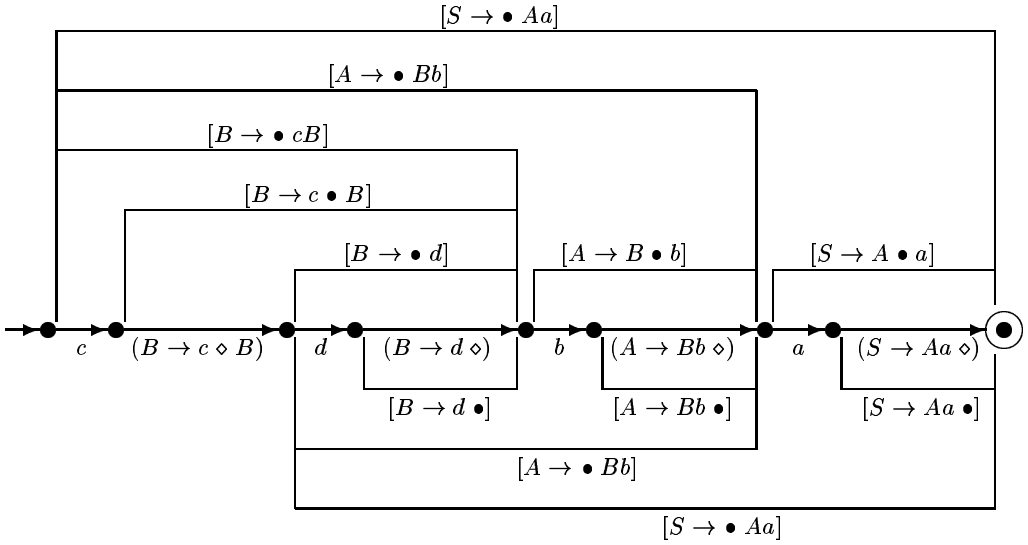

Figure 4. A finite automaton resulting from simulating the transducer on input *cdba* (thick lines), and the subsequent table U of dotted items (thin lines).

5 Retrieving a Parse Forest

Using the compact representation of all possible output strings discussed above, we can obtain the structure of the input according to the context-free grammar; by "structure" of the input we mean the collection of all parse trees. Again, we use a tabular representation, called a *parse forest* [6, 10, 2].

Our particular kind of parse forest is a table U consisting of *dotted items* of the form $[q, A \to \alpha \bullet \beta, q']$, where q and q' are states from K' and $A \to \alpha\beta$ is a rule. The dot indicates to how far recognition of the right-hand side has progressed. To be more precise, the meaning of the above dotted item is that the input symbols on a path from q to q' can be derived from β. Note that recognition of right-hand sides is done from right to left, i.e. in reversed order with respect to Earley's algorithm [6].

For a certain instance of a rule, the initial position of the dot is given by the position of the diamond in the corresponding filter item.

There are several ways to construct U. For presentational reasons our algorithm will be relatively simple, in the style of the CYK algorithm [8]:

1. Initially U is empty.
2. We perform one of the following until no more new elements can be added to U:
   (a) We choose a transition $(q, A \to \alpha \diamond, q') \in \Delta'$ and add an item $[q, A \to \alpha \bullet, q']$ to U.
   (b) We choose a transition $(q, A \to \alpha \diamond B, q') \in \Delta'$ and an item $[q', B \to \bullet\, \gamma, q''] \in U$ and add an item $[q, A \to \alpha \bullet B, q'']$ to U.
   (c) We choose a transition $(q, a, q') \in \Delta'$ and an item $[q', A \to \alpha a \bullet \beta, q''] \in U$ and add an item $[q, A \to \alpha \bullet a\beta, q'']$ to U.
   (d) We choose a pair of items $[q, B \to \bullet\, \gamma, q'], [q', A \to \alpha B \bullet \beta, q''] \in U$ and add an item $[q, A \to \alpha \bullet B\beta, q'']$ to U.

Assume the grammar is $G = (\Sigma, N, P, S)$. The following is to be performed for each set $N_i \in \mathcal{N}$ such that $\mathit{recursive}(N_i) = \mathit{self}$.

1. Add the following nonterminals to N: $A^{\uparrow}_B$, $A^{\downarrow}_B$, $\overleftarrow{A}_B$ and $\overrightarrow{A}_B$ for all $A, B \in N_i$.
2. Add the following rules to P, for all $A, B, C, D, E \in N_i$:
   - $A \to A^{\uparrow}_A$;
   - $A^{\uparrow}_B \to \overleftarrow{A}_C Y_1 \ldots Y_m C^{\downarrow}_B$, for all $(C \to Y_1 \ldots Y_m) \in P$, with $Y_1, \ldots, Y_m \notin N_i$;
   - $A^{\downarrow}_B \to \overrightarrow{C}_A Y_1 \ldots Y_m E^{\uparrow}_B$, for all $(D \to \alpha C Y_1 \ldots Y_m E \beta) \in P$, with $Y_1, \ldots, Y_m \notin N_i$;
   - $A^{\downarrow}_B \to \overrightarrow{B}_A$;
   - $\overleftarrow{A}_B \to Y_1 \ldots Y_m \overleftarrow{C}_B$, for all $(A \to Y_1 \ldots Y_m C \beta) \in P$, with $Y_1, \ldots, Y_m \notin N_i$;
   - $\overleftarrow{B}_B \to \varepsilon$;
   - $\overrightarrow{A}_B \to \overrightarrow{C}_B Y_1 \ldots Y_m$, for all $(A \to \alpha C Y_1 \ldots Y_m) \in P$, with $Y_1, \ldots, Y_m \notin N_i$;
   - $\overrightarrow{B}_B \to \varepsilon$.
3. Remove from P the old rules of the form $A \to \alpha$, where $A \in N_i$.
4. Reduce the grammar.

Figure 5. Approximation by transforming the grammar.

The items produced for the running example are represented as the thin lines in Figure 4.

6 Approximating a Context-Free Language

Section 3 presented a sufficient condition for the generated language to be regular, and explained when this condition is violated. This suggests how to change an arbitrary grammar so that it will come to satisfy the condition.

The intuition is that the "unbounded communication" between the left and right sides of spines is broken. This is done by a transformation that operates separately on each set N_i such that $\mathit{recursive}(N_i) = \mathit{self}$, as indicated in Figure 5. After this, the grammar will be strongly regular.

Consider the grammar of palindromes in the left half of Figure 1. The approximation algorithm leads to the grammar in the right half. Figure 1 (b) shows the effect on the structure of parse trees. Note that the left sides of former spines are treated by the new nonterminal $\overleftarrow{S}_S$ and the right sides by the new nonterminal $\overrightarrow{S}_S$.

This example deals with the special case that each nonterminal can lead to at most one recursive call of itself. The general case is more complicated and is treated elsewhere [15].

7 Obtaining Correct Parse Trees

In Section 5 we discussed how the table resulting from simulating the transducer should be interpreted in order to obtain a parse forest. However, we assumed then that the transducer had been constructed from a grammar that was strongly regular. In case the original grammar is *not* strongly regular we have to approach this task in a different way.

One possibility is to first apply the grammar transformation from the previous section and subsequently perform the 2-phase process as before. However, this approach results in a parse forest that reflects the structure of the transformed grammar rather than that of the original grammar.

The second and preferred approach is to incorporate the grammar transformation into the construction of the transducer. The accepted language is then the same as in the case of the first approach, but the symbols that occur in the output carry information about the rules from the *original* grammar.

How the construction of the finite transducer from Figure 2 needs to be changed is indicated in Figure 6. We only show the part of the code which deals with the case that α consists of a single nonterminal.

For nonterminals which are not in a set N_i such that $\mathit{recursive}(N_i) = \mathit{self}$, the same treatment as before is applied. Upon encountering a nonterminal $B \in N_i$ such that $\mathit{recursive}(N_i) = \mathit{self}$, we consider the structure of the grammar if it is transformed according to Figure 5. This transformation creates new sets of recursive nonterminals, which have to be treated according to Figure 2 depending on whether they may be left-recursive or right-recursive.

For example, given a fixed nonterminal $B \in N_i$, for some i such that $\mathit{recursive}(N_i) = \mathit{self}$, the set of nonterminals $A^{\uparrow}_B$ and $A^{\downarrow}_B$, for any $A \in N_i$, together form a set M in the transformed grammar for which $\mathit{recursive}(M) = \mathit{right}$. We may therefore construct the transducer as dictated by Figure 2 for this case. In particular, this relates to the rules of the form $A^{\uparrow}_B \to \overleftarrow{A}_C\, Y_1 \ldots Y_m C^{\downarrow}_B$, $A^{\downarrow}_B \to \overrightarrow{C}_A Y_1 \ldots Y_m E^{\uparrow}_B$ and $A^{\downarrow}_B \to \overrightarrow{B}_A$.

Note that a nonterminal of the form $\overleftarrow{A}_C$ does not belong to M but to another set, say M_1, which in the transformed grammar satisfies $\mathit{recursive}(M_1) = \mathit{right}$ (or $\mathit{recursive}(M_1) = \mathit{cyclic}$). Similarly, a nonterminal of the form $\overrightarrow{C}_A$ belongs to a set, say M_2, which satisfies $\mathit{recursive}(M_2) = \mathit{left}$ (or $\mathit{recursive}(M_2) = \mathit{cyclic}$). Treatment of these nonterminals occurs in a deeper level of recursion of *make_fst*, and appears as separate cases in Figure 6.

It is important to remember that the sets N_i in Figure 6 always refer to the nature of recursion in the *original* grammar; the transformed grammar is merely implicit in the given construction of the transducer, and helps us to understand the construction in terms of Figure 2.

In addition to I_{init}, filter items from the following set are used:

$$I_{mid} = \{B \to \alpha \diamond C\beta \mid (B \to \alpha C\beta) \in P \wedge \exists i[\mathit{recursive}(N_i) = \mathit{self} \wedge B, C \in N_i]\}$$

The meaning of the diamond is largely unchanged with regard to Section 3. For example, for the rule $D \to \alpha C Y_1 \ldots Y_m E\beta$, which corresponds to the rule $A^{\downarrow}_B \to \overrightarrow{C}_A\, Y_1 \ldots Y_m E^{\uparrow}_B$ of the transformed grammar, the filter item $D \to \alpha C Y_1 \ldots Y_m \diamond E\beta$ is output, which indicates that an instance of $Y_1 \ldots Y_m$ (or an approximation thereof) has just been read, which is potentially preceded by an instance of αC and followed by an instance of $E\beta$. On the other hand, upon encountering a rule such as $A^{\downarrow}_B \to \overrightarrow{B}_A$, which is an artifact of the grammar transformation, no output symbol is generated.

For retrieving the forest from $\mathcal{F}$ we need to take into account the additional form of filter item. Now the following steps are required:

(a) We choose $(q, A \to \alpha \diamond, q') \in \Delta'$ and add $[q, A \to \alpha \bullet, q']$ to U.
(b) We choose $(q, A \to \alpha \diamond B, q') \in \Delta'$, such that $(A \to \alpha \diamond B) \in I_{init}$, and $[q', B \to \bullet\, \gamma, q''] \in U$ and add $[q, A \to \alpha \bullet B, q'']$ to U.

...**else** (* α must consist of a single nonterminal *)
if α is of form $A \in N_i$, **some** i, **and** $recursive(N_i) \in \{right, left, cyclic\}$
then ...treatment as in Figure 2...
elseif α is of form $B \in N_i$, **some** i, **and** $recursive(N_i) = self$
then (* we implicitly replace B by $B^{\uparrow}_{B}$ according to $B \to B^{\uparrow}_{B}$ *)
for each $A \in N_i$ **do let** $q_{A^{\uparrow}_{B}} = fresh\_state$, $q_{A^{\downarrow}_{B}} = fresh\_state$ **end**;
for each $A \in N_i$ **and** $(C \to Y_1 \ldots Y_m) \in P$ **such that** $C \in N_i \wedge Y_1, \ldots, Y_m \notin N_i$
do let $q = fresh\_state$; $make\_fst(q_{A^{\uparrow}_{B}}, \overleftarrow{A}_C\ Y_1 \ldots Y_m, q)$;
let $\Delta = \Delta \cup \{(q, \varepsilon | (C \to Y_1 \ldots Y_m \diamond), q_{C^{\downarrow}_{B}})\}$
end; (* for $A^{\uparrow}_{B} \to \overleftarrow{A}_C\, Y_1 \ldots Y_m C^{\downarrow}_{B}$ *)
for each $A \in N_i$ **and** $(D \to \alpha C Y_1 \ldots Y_m E \beta) \in P$
such that $C, D, E \in N_i \wedge Y_1, \ldots, Y_m \notin N_i$
do let $q = fresh\_state$; $make\_fst(q_{A^{\downarrow}_{B}}, \overrightarrow{C}_A\ Y_1 \ldots Y_m, q)$;
let $\Delta = \Delta \cup \{(q, \varepsilon | (D \to \alpha C Y_1 \ldots Y_m \diamond E\beta), q_{E^{\uparrow}_{B}})\}$
end; (* for $A^{\downarrow}_{B} \to \overrightarrow{C}_A\, Y_1 \ldots Y_m E^{\uparrow}_{B}$ *)
for each $A \in N_i$
do $make\_fst(q_{A^{\downarrow}_{B}}, \overrightarrow{B}_A, q_1)$ (* for $A^{\downarrow}_{B} \to \overrightarrow{B}_A$ *)
end;
let $\Delta = \Delta \cup \{(q_0, \varepsilon | \varepsilon, q_{B^{\uparrow}_{B}})\}$
elseif $\overleftarrow{D}_B$ **such that** $D, B \in N_i$, **some** i
then for each $A \in N_i$ **do let** $q_{\overleftarrow{A_B}} = fresh\_state$ **end**;
for each $(A \to Y_1 \ldots Y_m C \beta) \in P$ **such that** $A, C \in N_i \wedge Y_1, \ldots, Y_m \notin N_i$
do let $q = fresh\_state$; $make\_fst(q_{\overleftarrow{A_B}}, Y_1 \ldots Y_m, q)$;
let $\Delta = \Delta \cup \{(q, \varepsilon | (A \to Y_1 \ldots Y_m \diamond C\beta), q_{\overleftarrow{C_B}})\}$
end; (* for $\overleftarrow{A}_B \to Y_1 \ldots Y_m \overleftarrow{C}_B$ *)
let $\Delta = \Delta \cup \{(q_{\overleftarrow{B_B}}, \varepsilon | \varepsilon, q_1)\}$; (* for $\overleftarrow{B}_B \to \varepsilon$ *)
let $\Delta = \Delta \cup \{(q_0, \varepsilon | \varepsilon, q_{\overleftarrow{D_B}})\}$
elseif $\overrightarrow{D}_B$ **such that** $D, B \in N_i$, **some** i
then for each $A \in N_i$ **do let** $q_{\overrightarrow{A_B}} = fresh\_state$ **end**;
for each $(A \to \alpha C Y_1 \ldots Y_m) \in P$ **such that** $A, C \in N_i \wedge Y_1, \ldots, Y_m \notin N_i$
do let $q = fresh\_state$; $make\_fst(q_{\overrightarrow{C_B}}, Y_1 \ldots Y_m, q)$;
let $\Delta = \Delta \cup \{(q, \varepsilon | (A \to \alpha C Y_1 \ldots Y_m \diamond), q_{\overrightarrow{A_B}})\}$
end; (* for $\overrightarrow{A}_B \to \overrightarrow{C}_B\, Y_1 \ldots Y_m$ *)
let $\Delta = \Delta \cup \{(q_0, \varepsilon | \varepsilon, q_{\overrightarrow{B_B}})\}$; (* for $\overrightarrow{B}_B \to \varepsilon$ *)
let $\Delta = \Delta \cup \{(q_{\overrightarrow{D_B}}, \varepsilon | \varepsilon, q_1)\}$
else let $A = \alpha$; (* α must consist of a single non-recursive nonterminal *)
...treatment as in Figure 2...
end

Figure 6. Code from Figure 2 changed to deal with arbitrary context-free grammars.

(c) We choose $(q, a, q') \in \Delta'$ and $[q', A \to \alpha a \bullet \beta, q''] \in U$ and add $[q, A \to \alpha \bullet a\beta, q'']$ to U.
(d) We choose $[q, B \to \bullet\ \gamma, q'], [q', A \to \alpha B \bullet \beta, q''] \in U$ and:
- if $(A \to \alpha \diamond B\beta) \in I_{mid}$, then add $[q''', A \to \alpha \bullet B\beta, q'']$ to U for each $(q''', A \to \alpha \diamond B\beta, q) \in \Delta'$, and
- otherwise, add $[q, A \to \alpha \bullet B\beta, q'']$ to U.

8 Empirical Results

The implementation was completed recently. Initial experiments allow some tentative conclusions, reported here.

We have compared the 2-phase algorithm to a traditional tabular context-free parsing algorithm. In order to allow a fair comparison, we have taken a mixed parsing strategy that applies a set of dotted items comparable to that of Section 7. Assuming the input is given by $a_1 \cdots a_n$ as before, the steps are given by:

(a) We choose i, such that $0 \leq i \leq n$, and $(A \to \alpha \diamond) \in I_{init}$ and add $[i, A \to \alpha \bullet, i]$ to U.
(b) We choose $[i, B \to \bullet\ \gamma, j] \in U$ and $(A \to \alpha \diamond B) \in I_{init}$, and add $[i, A \to \alpha \bullet B, j]$ to U.
(c) We choose $[i+1, A \to \alpha a_{i+1} \bullet \beta, j] \in U$ and add $[i, A \to \alpha \bullet a_{i+1}\beta, j]$ to U.
(d) We choose $[i, B \to \bullet\ \gamma, j], [j, A \to \alpha B \bullet \beta, k] \in U$ and add $[i, A \to \alpha \bullet B\beta, k]$ to U.

For the experiments we have taken a grammar for German, generated automatically through EBL, of which a considerable part contains self-embedding. The transducer was determinized and minimized as if it were a finite automaton, i.e. in a transition $(q, v|w, q')$ the pair $v|w$ is treated as one symbol, and the pair $\varepsilon|\varepsilon$ is treated as the empty string. The test sentences were obtained using a random generator [14].

For a given input sentence, we define T_1 and T_2 to be the number of steps that are performed for the respective phases of the 2-phase algorithm: first, the creation of $\mathcal{F}$ from the input $a_1 \cdots a_n$, and second, the creation of U from $\mathcal{F}$. We define T_{cf} to be the number of steps that are performed for the direct construction of table U from $a_1 \cdots a_n$ by the above tabular algorithm.

Concerning the two processes with context-free power, viz. T_{cf} and T_2, we have observed that in the majority of cases there is a reduction in the number of steps from T_{cf} to T_2. This can be a reduction from several hundreds of steps to less than 10. In individual cases however, especially for long sentences, T_2 can be larger than T_{cf}. This can be explained by the fact that $\mathcal{F}$ may have many more states than that the input sentence has positions, which leads to less sharing of computation.

Adding T_1 and T_2 in many cases leads to higher numbers of steps than T_{cf}. At this stage we cannot say whether this implies that the 2-phase idea is not useful. Many refinements, especially concerning the reduction of the number of states of $\mathcal{F}$ in order to enhance sharing of computation, have as yet not been explored.

In this context, we observe that the size of the repertoire of filter items has conflicting consequences for the overall complexity. If $\mathcal{T}$ outputs no filter items, then it reduces to a recognizer, which can be determinized. Consequently, T_1 will be equal to the sentence length, but T_2 will be no less than (and in fact identical to) T_{cf}. If on the other hand $\mathcal{T}$ outputs many types of filter item, then determinization and minimization is more difficult and consequently $\mathcal{F}$ may be large and both T_1 and T_2 may be high.

Acknowledgements

Parts of this research were carried out within the framework of the Priority Programme Language and Speech Technology (TST), while the author was employed at the University of Groningen. The TST-Programme is sponsored by NWO (Dutch Organization for Scientific Research). This work was further funded 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. The responsibility for the contents lies with the author.

References

1. J. Berstel. 1979. *Transductions and Context-Free Languages.* B.G. Teubner, Stuttgart.
2. S. Billot and B. Lang. 1989. The structure of shared forests in ambiguous parsing. In *27th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, pages 143–151, Vancouver, British Columbia, Canada, June.
3. N. Chomsky. 1959. A note on phrase structure grammars. *Information and Control*, 2:393–395.
4. N. Chomsky. 1959. On certain formal properties of grammars. *Information and Control*, 2:137–167.
5. K. Čulik II and R. Cohen. 1973. LR-regular grammars—an extension of LR(k) grammars. *Journal of Computer and System Sciences*, 7:66–96.
6. J. Earley. 1970. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102, February.
7. E. Grimley Evans. 1997. Approximating context-free grammars with a finite-state calculus. In *35th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, pages 452–459, Madrid, Spain, July.
8. M.A. Harrison. 1978. *Introduction to Formal Language Theory.* Addison-Wesley.
9. S. Krauwer and L. des Tombe. 1981. Transducers and grammars as theories of language. *Theoretical Linguistics*, 8:173–202.
10. B. Lang. 1974. Deterministic techniques for efficient non-deterministic parsers. In *Automata, Languages and Programming, 2nd Colloquium*, Lecture Notes in Computer Science, volume 14, pages 255–269, Saarbrücken. Springer-Verlag.
11. D.T. Langendoen. 1975. Finite-state parsing of phrase-structure languages and the status of readjustment rules in grammar. *Linguistic Inquiry*, 6(4):533–554.
12. D.T. Langendoen and Y. Langsam. 1990. A new method of representing constituent structures. *Annals New York Academy of Sciences*, 583:143–160.
13. M. Mohri. 1997. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311.
14. M.-J. Nederhof. 1996. Efficient generation of random sentences. *Natural Language Engineering*, 2(1):1–13.
15. M.-J. Nederhof. 1997. Regular approximations of CFLs: A grammatical view. In *International Workshop on Parsing Technologies*, pages 159–170, Massachusetts Institute of Technology, September.
16. F.C.N. Pereira and R.N. Wright. 1997. Finite-state approximation of phrase-structure grammars. In E. Roche and Y. Schabes, editors, *Finite-State Language Processing*, pages 149–173. MIT Press.
17. D.J. Rosenkrantz and P.M. Lewis II. 1970. Deterministic left corner parsing. In *IEEE Conference Record of the 11th Annual Symposium on Switching and Automata Theory*, pages 139–152.