Chapter 1

REGULAR APPROXIMATION OF CFLS: A GRAMMATICAL VIEW

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

Abstract We show that for each context-free grammar a new grammar can be constructed that generates a regular language. This construction differs from some existing methods of approximation in that use of a pushdown automaton is avoided. This allows better insight into how the generated language is affected.

Introduction

In existing literature, a number of methods have been proposed for approximating a context-free language (CFL) by means of a regular language. Some of these methods were expressed in terms of 1) a construction of a pushdown automaton from the grammar, where the language accepted by the automaton is identical to the language generated by the grammar, and 2) an approximation of that pushdown automaton, expressed in terms of a finite automaton.

Pushdown automata manipulate stacks, and the set of different stacks that may need to be considered for processing of different input strings is potentially infinite, and due to this fact, a pushdown automaton may accept a language that is not regular. The approximation process now consists in reducing the infinite set of stacks that are distinguished by the pushdown automaton to a finite set, and thus a finite automaton results. As will be explained later, there are roughly two ways of achieving this, the first leading to regular languages that are subsets of the original context-free languages (Krauwer and des Tombe, 1981; Langendoen and Langsam, 1987; Pulman, 1986; Johnson, 1998), the

second leading instead to supersets (Baker, 1981; Bermudez and Schimpf, 1990; Pereira and Wright, 1997).

A disadvantage of this kind of approximation is that it is difficult to understand or to influence how a language is changed in the process. This holds in particular for methods of approximation that make use of a nontrivial construction of pushdown automata from grammars. An example is the construction of LR recognizers (Sippu and Soisalon-Soininen, 1990). The structure of a grammar is very different from the structure of the LR automaton constructed from it. How subsequent manipulation of the LR automaton affects the language in terms of the grammar is very difficult to see, and there seems to be no obvious way to make adjustments to the approximation process.

In this communication we present an approximation that avoids the use of pushdown automata altogether, and which can be summarized as follows. 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.

The structure of this paper is as follows. In Section 1. we recall some standard definitions from formal language theory. Section 2. investigates a sufficient condition for a context-free grammar to generate a regular language. An algorithm to transform a grammar such that this condition is satisfied is given in Section 3..

As Section 4. shows, some aspects of our method are undecidable. A refinement for obtaining more precise approximations is presented in Section 5.. Section 6. compares our method to other methods. Conclusions are found in Section 7..

1. 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 G 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 $w \in \Sigma^*$.

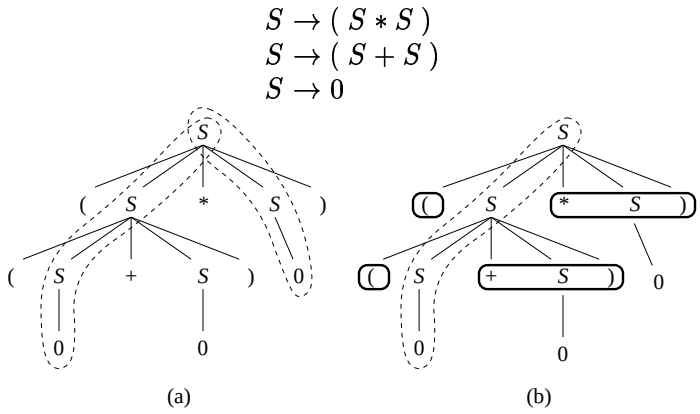

Figure 1.1 (a) two spines in a parse tree, (b) the grammar symbols to the left and right of a spine immediately dominated by nodes on the spine.

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 rule of the form $A \to \epsilon$ is called an *epsilon* rule. A grammar is called *cyclic* if $A \to^* A$, for some A.

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 $(s, v) \vdash^* (q, \epsilon)$, 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.

2. 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. Figure 1.1 (a) indicates two spines in a parse tree for the string $((0 + 0) * 0)$, according to a simple grammar.

Our main interest in spines lies in the sequences of grammar symbols at nodes bordering on spines. Figure 1.1 (b) gives an example: to the left of the spine we find the sequence "((" and to the right we find "$+S) * S)$". The way that such pairs of sequences may relate to each other determines the strength of

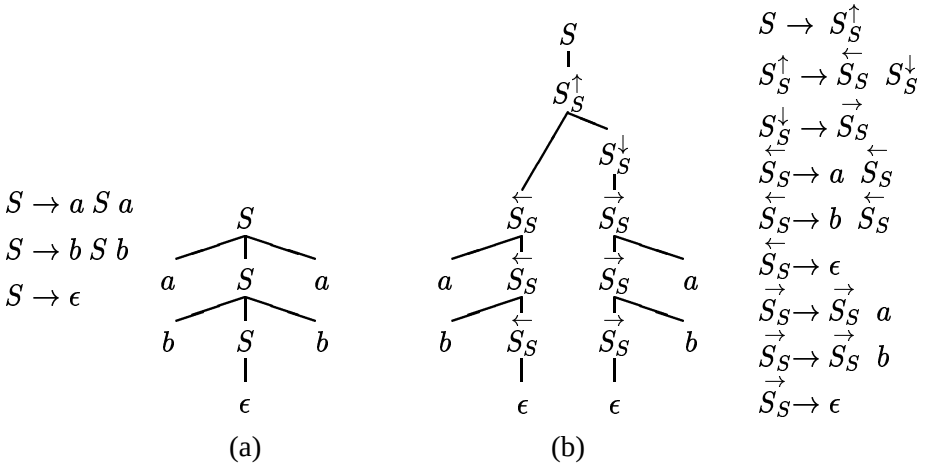

Figure 1.2 Parse trees for a palindrome: (a) original grammar, (b) transformed grammar (Section 3.).

context-free grammars and, as we will see later, by restricting this relationship we may reduce the generative power of context-free grammars to the regular languages.

A simpler example is the set of parse trees such as the one in Figure 1.2 (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 number 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, due to (Chomsky, 1959b):

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

In order to avoid the somewhat unfortunate term *nonself-embedding* (or *noncenter-embedding*, as in (Langendoen, 1975)) 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.

Obviously, right linear and left linear grammars (as known from standard literature such as (Hopcroft and Ullman, 1979)) are strongly regular. That right

linear and left linear grammars generate regular languages is easy to show. That strongly regular grammars also generate regular languages will be proved shortly.

First, for an arbitrary grammar, we define the set of *recursive* nonterminals as:

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

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

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

We now define the function *recursive* from $\mathcal{N}$ to the set $\{$*left*, *right*, *self*, *cyclic*$\}$. For $1 \leq i \leq n$:

$$\begin{array}{rcll}
\mathit{recursive}(N_i) & = & \mathit{left}, & \mathit{if} \;\; \neg \mathit{LeftGenerating}(N_i) \;\; \wedge \\
 & & & \quad \mathit{RightGenerating}(N_i) \\
 & = & \mathit{right}, & \mathit{if} \;\; \mathit{LeftGenerating}(N_i) \;\; \wedge \\
 & & & \quad \neg \mathit{RightGenerating}(N_i) \\
 & = & \mathit{self}, & \mathit{if} \;\; \mathit{LeftGenerating}(N_i) \;\; \wedge \\
 & & & \quad \mathit{RightGenerating}(N_i) \\
 & = & \mathit{cyclic}, & \mathit{if} \;\; \neg \mathit{LeftGenerating}(N_i) \;\; \wedge \\
 & & & \quad \neg \mathit{RightGenerating}(N_i)
\end{array}$$

where

$$\begin{aligned}
\mathit{LeftGenerating}(N_i) &= \exists (A \rightarrow \alpha B \beta) \in P [A \in N_i \wedge B \in N_i \wedge \alpha \neq \epsilon] \\
\mathit{RightGenerating}(N_i) &= \exists (A \rightarrow \alpha B \beta) \in P [A \in N_i \wedge B \in N_i \wedge \beta \neq \epsilon]
\end{aligned}$$

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. We only prove this in one direction: Suppose we have two rules $A_1 \rightarrow \alpha_1 B_1 \beta_1$, $A_1, B_1 \in N_i$, and $A_2 \rightarrow \alpha_2 B_2 \beta_2$, $A_2, B_2 \in N_i$, such that $\alpha_1 \neq \epsilon$ and $\beta_2 \neq \epsilon$. This means that $A_1 \rightarrow \alpha_1 B_1 \beta_1 \rightarrow^* \alpha_1 \alpha_1' A_2 \beta_1' \beta_1 \rightarrow \alpha_1 \alpha_1' \alpha_2 B_2 \beta_2 \beta_1' \beta_1 \rightarrow^* \alpha_1 \alpha_1' \alpha_2 \alpha_2' A_1 \beta_2' \beta_2 \beta_1' \beta_1$, for some

$\alpha'_1, \beta'_1, \alpha'_2, \beta'_2$, making use of the assumption that B_1 and A_2, and then B_2 and A_1 are in the same subset N_i of mutually recursive nonterminals. In the final sentential form we have $\alpha_1\alpha'_1\alpha_2\alpha'_2 \neq \epsilon$ and $\beta'_2\beta_2\beta'_1\beta_1 \neq \epsilon$, and therefore the grammar is self-embedding.

A set N_i such that $recursive(N_i) = self$ thus provides an isolated aspect of the grammar that causes self-embedding, and therefore making the grammar strongly regular will depend on a solution for how to transform the part of the grammar in which nonterminals from N_i occur.

We now prove that a grammar that is strongly regular (or in other words, for all i, $recursive(N_i) \in \{left, right, cyclic\}$) generates a regular language. Our proof differs from a proof of the same fact in (Chomsky, 1959a) in that it is fully constructive: Figure 1.3 presents an algorithm for creating a finite automaton that accepts the language generated by the grammar.

The process is initiated at the start symbol, and from there the process descends the grammar in all ways until terminals are encountered, and then transitions are created labelled with those terminals. Descending the grammar is straightforward in the case of rules of which the left-hand side is not a recursive nonterminal: the groups of transitions 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 1.3 cyclic and right-recursive subsets N_i are treated uniformly.

We discuss the case that the nonterminals are left-recursive. (The converse case is left to the imagination of the reader.) 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 (Rosenkrantz and Lewis II, 1970), and specifically of one construction that focuses on groups of mutually recursive nonterminals (Nederhof, 1994a, Section 5.8).

An example is given in Figure 1.4. Four states have been labelled according to the names they are given in procedure *make_fa*. 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.

3. APPROXIMATING A CONTEXT-FREE LANGUAGE

Now that we know what makes a context-free grammar violate a sufficient condition for the generated language to be regular, we have a good starting point to investigate how we should change a grammar in order to obtain a regular

let $K = \emptyset$, $\Delta = \emptyset$, s = *fresh_state*, f = *fresh_state*, $F = \{f\}$;
make_fa(s, S, f).

procedure *make_fa*(q_0, α, q_1):
 if $\alpha = \epsilon$
 then let $\Delta = \Delta \cup \{(q_0, \epsilon, q_1)\}$
 elseif $\alpha = a$, **some** $a \in \Sigma$
 then let $\Delta = \Delta \cup \{(q_0, a, q_1)\}$
 elseif $\alpha = X\beta$, **some** $X \in V$, $\beta \in V^*$ **such that** $|\beta| > 0$
 then let q = *fresh_state*;
 make_fa(q_0, X, q);
 make_fa(q, β, 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 = *fresh_state* **end**;
 if *recursive*(N_i) = *left*
 then for each $(C \to X_1 \ldots X_m) \in P$ **such that**
 $C \in N_i \wedge X_1, \ldots, X_m \notin N_i$
 do *make_fa*$(q_0, X_1 \ldots X_m, q_C)$
 end;
 for each $(C \to DX_1 \ldots X_m) \in P$ **such that**
 $C, D \in N_i \wedge X_1, \ldots, X_m \notin N_i$
 do *make_fa*$(q_D, X_1 \ldots X_m, q_C)$
 end;
 let $\Delta = \Delta \cup \{(q_A, \epsilon, q_1)\}$
 else "the converse of the then-part"
 (* *recursive*$(N_i) \in \{$*right*, *cyclic*$\}$ *)
 end
 else for each $(A \to \beta) \in P$ **do** *make_fa*(q_0, β, q_1) **end**
 (* A is not recursive *)
 end
 end
end.

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

Figure 1.3 Mapping from a strongly regular grammar $G = (\Sigma, N, P, S)$ into an equivalent finite automaton $\mathcal{F} = (K, \Sigma, \Delta, s, F)$.

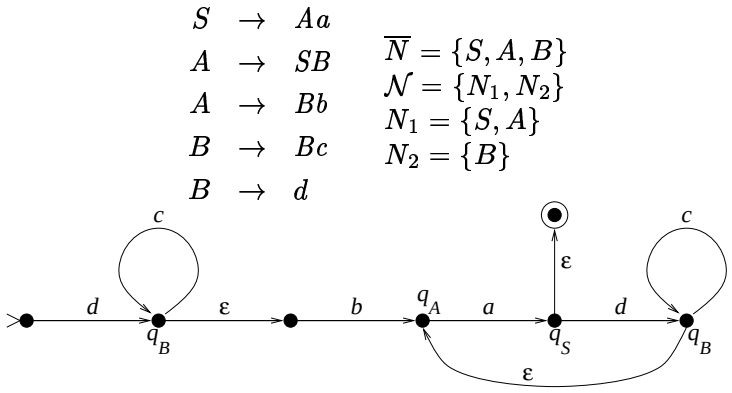

Figure 1.4 Application of the code from Figure 1.3 on a small grammar.

language. The intuition is that the "unbounded communication" between the left and right sides of spines is broken.

We concentrate on the sets N_i with *recursive*(N_i) = *self*. For each set separately, we apply the transformation in Figure 1.5. Thereafter the grammar will be strongly regular. We will explain the transformation by means of two examples.

We first discuss the special case that each nonterminal can lead to at most one recursive call of itself, which holds for *linear* context-free grammars (Hopcroft and Ullman, 1979). Consider the grammar of palindromes in the left half of Figure 1.2. The approximation algorithm leads to the grammar in the right half. Figure 1.2 (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 $\overrightarrow{S_S}$.

The general case is more complicated. A nonterminal A may lead to several recursive occurrences: $A \rightarrow^* \alpha A \beta A \gamma$. As before, our approach is to approximate the language by separating the left and right sides of spines, but in this case, several spines in a single parse tree may need to be taken care of at once.

As a presentation of this case in a pictorial way, Figure 1.6 (a) suggests a part of a parse tree in which all (labels of the) nodes belong to the same set N_i, where *recursive*(N_i) = *self*. Other nodes in the direct vicinity of the depicted part of the parse tree we assume not to be in N_i; the triangles $\triangle$, for example, denote a mother node in N_i and a number of daughter nodes *not* in N_i. The dotted lines labelled $p1, p3, p5, p7$ represent paths along nodes in N_i such that the nodes to the left of the visited nodes are not in N_i. In the case of $p2, p4, p6, p8$ the nodes to the right of the visited nodes are not in N_i.

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

1. Add the following nonterminals to N: $A^{\uparrow}_B$, $A^{\downarrow}_B$, $\overleftarrow{A}_B$ and $\overrightarrow{A}_B$ for all pairs of $A, B \in N_i$.
2. Add the following rules to P, for all $A, B, C, D, E \in N_i$:
   - $A \to A^{\uparrow}_A$;
   - $A^{\uparrow}_B \to \overleftarrow{A}_C \, X_1 \ldots X_m C^{\downarrow}_B$, for all $(C \to X_1 \ldots X_m) \in P$, with $X_1, \ldots, X_m \notin N_i$;
   - $A^{\downarrow}_B \to \overrightarrow{C}_A \, X_1 \ldots X_m E^{\uparrow}_B$, for all $(D \to \alpha C X_1 \ldots X_m E \beta) \in P$, with $X_1, \ldots, X_m \notin N_i$;
   - $A^{\downarrow}_B \to \overrightarrow{B}_A$;
   - $\overleftarrow{A}_B \to X_1 \ldots X_m \, \overleftarrow{C}_B$, for all $(A \to X_1 \ldots X_m C \beta) \in P$, with $X_1, \ldots, X_m \notin N_i$;
   - $\overleftarrow{A}_A \to \epsilon$;
   - $\overrightarrow{A}_B \to \overrightarrow{C}_B \, X_1 \ldots X_m$, for all $(A \to \alpha C X_1 \ldots X_m) \in P$, with $X_1, \ldots, X_m \notin N_i$;
   - $\overrightarrow{A}_A \to \epsilon$.
3. Remove from P the old rules of the form $A \to \alpha$, where $A \in N_i$.
4. Reduce the grammar.

Figure 1.5 Approximation by transforming the grammar, given a set N_i.

The effect of our transformation on the structure of the parse tree is suggested in Figure 1.6 (b). We see that the left and right sides of spines (e.g. $p1$ and $p2$) are disconnected, in the same way as "unbounded communication" between the two sides of spines was broken in our earlier example of the palindrome grammar.

Consider now the following grammar for mathematical expressions:

$$\begin{aligned} S &\to A * B \\ A &\to (A + B) \end{aligned}$$

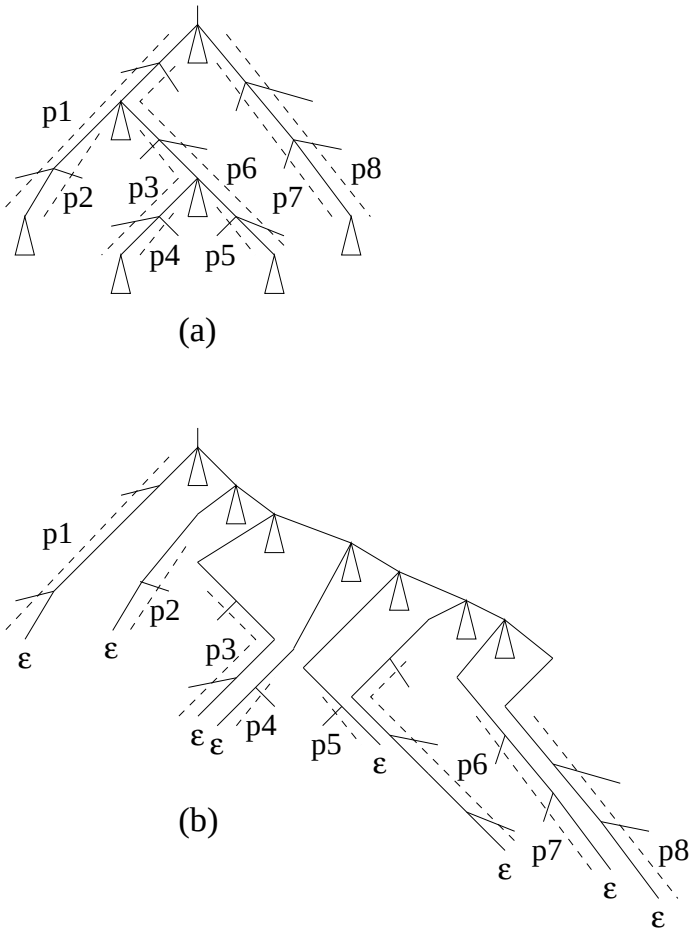

Figure 1.6 The general effect of the transformation on the structure of parse trees.

$$\begin{aligned} A &\rightarrow a \\ B &\rightarrow [A] \\ B &\rightarrow b \end{aligned}$$

We have $\overline{N} = \{A, B\}$, $\mathcal{N} = \{N_1\}$, $N_1 = \{A, B\}$, and *recursive*$(N_1) =$ *self*. After applying the approximation algorithm to N_1 we obtain the grammar in Figure 1.7. We emphasize that the transformed grammar in the figure has already been reduced.

If we compare this example to the general picture in Figure 1.6 (b), we conclude that a nonterminal such as $\overleftarrow{B}_A$ derives paths such as $p1, p3, p5$ or $p7$, where B was the top label at the path in the original parse tree and A occurred

$$
\begin{array}{llll}
S \to A * B & A^{\downarrow}_{A} \to \overrightarrow{A}_{A} \; + B^{\uparrow}_{A} & \overleftarrow{A}_{A} \to (\; \overleftarrow{A}_{A} & \overrightarrow{A}_{A} \to \overrightarrow{B}_{A} \;) \\
A \to A^{\uparrow}_{A} & B^{\downarrow}_{A} \to \overrightarrow{A}_{B} \; + B^{\uparrow}_{A} & \overleftarrow{B}_{A} \to [\; \overleftarrow{A}_{A} & \overrightarrow{A}_{B} \to \overrightarrow{B}_{B} \;) \\
B \to B^{\uparrow}_{B} & A^{\downarrow}_{B} \to \overrightarrow{A}_{A} \; + B^{\uparrow}_{B} & \overleftarrow{A}_{A} \to \epsilon & \overrightarrow{B}_{A} \to \overrightarrow{A}_{A} \;] \\
A^{\uparrow}_{A} \to \overleftarrow{A}_{A} \; a \; A^{\downarrow}_{A} & B^{\downarrow}_{B} \to \overrightarrow{A}_{B} \; + B^{\uparrow}_{B} & \overleftarrow{B}_{B} \to \epsilon & \overrightarrow{B}_{B} \to \overrightarrow{A}_{B} \;] \\
B^{\uparrow}_{A} \to \overleftarrow{B}_{A} \; a \; A^{\downarrow}_{A} & A^{\downarrow}_{A} \to \overrightarrow{A}_{A} & & \overrightarrow{A}_{A} \to \epsilon \\
B^{\uparrow}_{B} \to \overleftarrow{B}_{A} \; a \; A^{\downarrow}_{B} & B^{\downarrow}_{A} \to \overrightarrow{A}_{B} & & \overrightarrow{B}_{B} \to \epsilon \\
B^{\uparrow}_{A} \to \overleftarrow{B}_{B} \; b \; B^{\downarrow}_{A} & A^{\downarrow}_{B} \to \overrightarrow{B}_{A} & & \\
B^{\uparrow}_{B} \to \overleftarrow{B}_{B} \; b \; B^{\downarrow}_{B} & B^{\downarrow}_{B} \to \overrightarrow{B}_{B} & &
\end{array}
$$

Figure 1.7 Transformed grammar, resulting from application of Figure 1.5.

at the bottom. A similar fact holds for nonterminals such as $\overrightarrow{B}_{A}$. Nonterminals such as $B^{\uparrow}_{A}$ and $B^{\downarrow}_{A}$ indicate that the root of the complete subtree was labelled A, and that the last node of the tree that was treated is labelled B; in the case of $B^{\uparrow}_{A}$ that node is at the top of a path such as $p1, p3, p5$ or $p7$ in the original tree, in the case of $B^{\downarrow}_{A}$ that node is at the bottom of a path such as $p2, p4, p6$ or $p8$.

4. LIMITATIONS

It is undecidable whether the language generated by a context-free grammar is regular (Harrison, 1978). Consequently, the condition of strong regularity, which is decidable and is a sufficient condition for the language to be regular, cannot also be a necessary condition. This is demonstrated by the following grammar:[1]

$$
\begin{array}{lcl}
S & \to & a \; A \mid B \; a \mid C \\
A & \to & a \; A \mid C \\
B & \to & B \; a \mid C \\
C & \to & a \; C \; a \mid c
\end{array}
$$

This (non-ambiguous) grammar generates the regular language $a^{*}ca^{*}$. Yet it is not strongly regular, due to the cycle pertaining to the rule $C \to a \; C \; a$. Fortunately, our algorithm transforms this grammar into a strongly regular one which generates the same language $a^{*}ca^{*}$.

In some cases, a grammar that is not strongly regular but that generates a regular language is transformed into one that generates a strictly larger language. This cannot be prevented however by any method of superset approximation, since transforming a context-free grammar generating a regular language into a finite automaton accepting the same language is an unsolvable problem (Ullian, 1967).

A simple example where our method has this undesirable behaviour is the following:

$$S \rightarrow aSa \mid aSb \mid bSa \mid bSb \mid \epsilon$$

This grammar generates the regular language of all strings over $\{a, b\}$ of even length. Our approximation however results in the exact same grammar as in the example of palindromes. This grammar generates the regular language of *all* strings over $\{a, b\}$ — not only those of even length.[2]

If we represent context-free languages by means of pushdown automata (see also Section 6.), we can define a subclass for which regularity is decidable, namely those that allow a deterministic pushdown automaton. If such a deterministic language is regular, we can furthermore construct an equivalent deterministic finite automaton (Stearns, 1967). It turns out that even for this restricted class of context-free languages, the construction of equivalent finite automata is quite complex: The number of states of the (deterministic) finite automata may be a double exponential function in the size of the original deterministic pushdown automata (Meyer and Fischer, 1971; Valiant, 1975).

For arbitrary context-free grammars that generate regular languages, no recursive function in the size of grammars exists that provides an upper bound to the number of states of equivalent finite automata (Meyer and Fischer, 1971).

5. REFINEMENT

Our approximation algorithm is such that the two sides of spines are disconnected for all nonterminals that are involved in self-embedding (i.e. those in some fixed N_i with $\mathit{recursive}(N_i) = \mathit{self}$). One can however retain a finite amount of self-embedding in a precise way by unfolding j levels of applications of rules before the approximation algorithm is applied. In the unfolded rules, recursive nonterminals are replaced by new non-recursive nonterminals, so that in those j levels the precision remains unaffected.

One way of achieving this is the following. For each nonterminal $A \in N_i$ we introduce j fresh nonterminals $A[1], \ldots, A[j]$, and for each $A \rightarrow X_1 \cdots X_m$ in P such that $A \in N_i$, and h such that $1 \leq h \leq j$, we add $A[h] \rightarrow X'_1 \cdots X'_m$ to P, where

$$\begin{aligned} X'_k &= X_k[h+1], \text{ if } X_k \in N_i \wedge h < j \\ &= X_k, \text{ otherwise} \end{aligned}$$

Further, we replace all rules $A \rightarrow X_1 \cdots X_m$ such that $A \notin N_i$ by $A \rightarrow X'_1 \cdots X'_m$, where

$$\begin{aligned} X'_k &= X_k[1], \text{ if } X_k \in N_i \\ &= X_k, \text{ otherwise} \end{aligned}$$

If the start symbol S was in N_i, we let $S[1]$ be the new start symbol. Note that the transformation preserves the language.

If we take $j = 3$, the palindrome grammar becomes:

$$\begin{array}{rcl} S[1] & \rightarrow & a\ S[2]\ a \mid b\ S[2]\ b \mid \epsilon \\ S[2] & \rightarrow & a\ S[3]\ a \mid b\ S[3]\ b \mid \epsilon \\ S[3] & \rightarrow & a\ S\ a \mid b\ S\ b \mid \epsilon \\ S & \rightarrow & a\ S\ a \mid b\ S\ b \mid \epsilon \end{array}$$

After applying the approximation algorithm, all generated strings up to length 6 are palindromes. Only generated strings longer than 6 may not be palindromes: these are of the form $wvv'w^R$, for some $w \in \{a,b\}^3$ and $v, v' \in \{a,b\}^*$, where w^R indicates the mirror image of w. Thus the outer 3 symbols left and right do match, but not the innermost symbols in both "halves".

In general, by choosing j high enough we can obtain approximations that are language-preserving up to a certain string length, provided however the grammar is not cyclic. Apart from actual cyclic grammars, the above grammar transformation becomes less effective when the original grammar contains many unit rules or epsilon rules, which can be explained by the fact that such rules do not contribute to the length of the generated string. This problem can be overcome by eliminating such rules from the grammar before applying the above transformation.

The transformation above in effect decorates nodes in the parse tree with numbers up to j indicating the distance to the nearest ancestor node not in N_i. The second refinement we discuss has the effect of indicating the distance up to j to the furthest descendent not in N_i.

For this refinement, we again introduce j fresh nonterminals $A[1], \ldots, A[j]$ for each $A \in N_i$. If the start symbol S is in N_i, we first introduce a new nonterminal $S^\dagger$, which is to become the new start symbol and we add the rule $S^\dagger \rightarrow S$. Then, each $A \rightarrow X_1 \cdots X_m$ in P is replaced by a collection of other rules, which are created as follows. We determine the (possibly empty) list $k_1, \ldots, k_p$ of ascending indices of members that are in N_i: $\{k_1, \ldots, k_p\} = \{k \mid 1 \leq k \leq m \wedge X_k \in N_i\}$ and $k_1 < \ldots < k_p$. For each list of p numbers $n_1, \ldots, n_p \in \{1, \ldots, j, j+1\}$ we create the rule $A' \rightarrow X'_1 \cdots X'_m$, where

$$\begin{array}{rcl} X'_k & = & X_k[n_k], \text{ if } X_k \in N_i \wedge n_k \leq j \\ & = & X_k, \text{ otherwise} \\ A' & = & A[h+1], \text{ if } A \in N_i \wedge h < j, \text{ where } h = \max\limits_{1 \leq k \leq p} n_k \\ & = & A, \text{ otherwise} \end{array}$$

We assume that h evaluates to 0 if $p = 0$. Note that $j + 1$ is an auxiliary number which does not show up in the transformed grammar; it represents the case that the distance to the furthest descendent not in N_i is more than j. This transformation also preserves the language.

For the running example, with $j = 3$, we obtain:

$$\begin{array}{rcl} S^{\dagger} & \rightarrow & S \mid S[3] \mid S[2] \mid S[1] \\ S & \rightarrow & a\ S\ a \mid b\ S\ b \mid a\ S[3]\ a \mid b\ S[3]\ b \\ S[3] & \rightarrow & a\ S[2]\ a \mid b\ S[2]\ b \\ S[2] & \rightarrow & a\ S[1]\ a \mid b\ S[1]\ b \\ S[1] & \rightarrow & \epsilon \end{array}$$

Approximation now results in palindromes up to length 6, but longer strings have the form vww^Rv', for some $w \in \{a,b\}^3$ and $v, v' \in \{a,b\}^*$, where it is the innermost, not the outermost, part that still has the characteristics of palindromes.

6. OTHER METHODS OF APPROXIMATION

In existing literature, several methods of regular approximation of context-free languages are described in terms of the approximation of pushdown automata. A *pushdown automaton* $\mathcal{A}$ is a 5-tuple $(Q, \Sigma, \Delta, I, F)$, where Q is a finite set of *stack symbols*, of which I is the *initial* stack symbol and F is the *final* stack symbol, Σ is the input alphabet, and the *transition relation* Δ is a finite subset of $Q^* \times \Sigma^* \times Q^*$.

A configuration here is an element of $Q^* \times \Sigma^*$. The first components of such elements are called *stacks*. We define the binary relation $\vdash$ between configurations as: $(\alpha\beta, vw) \vdash (\alpha\gamma, w)$ if and only if $(\beta, v, \gamma) \in \Delta$. Some input v is *recognized* if $(I, v) \vdash^* (\alpha F, \epsilon)$, for some $\alpha \in Q^*$.

The language *accepted* by $\mathcal{A}$ is defined to be the set of all strings v that are recognized. A language is accepted by a pushdown automaton if and only if it is a context-free language. That every context-free language is accepted by a pushdown automaton is witnessed by a number of constructions of pushdown automata from context-free grammars. Let us call such a construction a *parsing strategy*; see (Nederhof, 1994b) for a family of parsing strategies.

In general, there is an infinite set of stacks α that satisfy $(I, v) \vdash^* (\alpha, \epsilon)$. Irrespective of the parsing strategy, we can define approximations in terms of operations that in effect reduce the infinite set of stacks that are distinguished by the pushdown automaton to a finite set, and thus lead to a finite automaton.

One such operation is simply a restriction of the relation $\vdash$ to a smaller relation $\vdash_d$ that disallows the stack height to exceed a fixed number $d \geq 1$. Formally, $(\alpha, vw) \vdash_d (\beta, w)$ if and only if $(\alpha, vw) \vdash (\beta, w)$ and $|\alpha| \leq d$ and $|\beta| \leq d$. One can now construct a finite automaton where the set K of states is defined to be the set of stacks α that satisfy $(I, v) \vdash_d^* (\alpha, \epsilon)$. The initial state is I, and the final states are stacks from K of the form αF. The transition relation Δ is defined to be $(\alpha, v, \beta) \in \Delta$ if and only if $(\alpha, v) \vdash_d (\beta, \epsilon)$.

This kind of subset approximation is proposed in (Krauwer and des Tombe, 1981; Langendoen and Langsam, 1987; Pulman, 1986; Johnson, 1998) in

combination with the (slightly modified) left-corner parsing strategy. The motivation for this strategy is that the approximation is then exact when the grammar is strongly regular, provided that d is chosen high enough. However, any other parsing strategy can be used as well, which may lead to different approximating languages.

Some theoretical limitations of subset approximation have been investigated by (Ullian, 1967): Given a context-free language, it is undecidable whether an infinite regular subset exists; yet, given that it exists, it can be computed. Note that for practical purposes one is interested in determining a "large" regular subset, not just any infinite subset of a context-free language as in the theorem from (Ullian, 1967). Experiments reported in (Nederhof, 2000) show that computing subset approximations may be very expensive for practical grammars.

Superset approximations result if we define a mapping f from the infinite set of stacks to a finite domain. We then construct a finite automaton of which K, the set of states, consists of the elements from that finite domain of the form $f(\alpha)$ such that α satisfies $(I, v) \vdash^* (\alpha, \epsilon)$, for some v. The initial state is defined to be $f(I)$ and the final states are the elements from K of the form $f(\alpha F)$, for stacks αF satisfying $(I, v) \vdash^* (\alpha F, \epsilon)$, for some v. The transition relation Δ of the finite automaton is defined to be the least relation such that $(I, w) \vdash^* (\alpha, \epsilon)$ and $(\alpha, v) \vdash (\beta, \epsilon)$ implies $(f(\alpha), v, f(\beta)) \in \Delta$.

As an example, let us consider a top-down parsing strategy, described as follows. We define Q to be the set of "dotted" rules of the form $[A \to \alpha \bullet \beta]$, where $(A \to \alpha\beta)$ in P. Let us further assume, without loss of generality, that there is only one occurrence of start symbol S, which is found in the (unique) rule of the form $S \to \sigma$. We then choose $I = [S \to \bullet\, \sigma]$ and $F = [S \to \sigma \bullet]$. The transition relation Δ is defined by the following:

- $([A \to \alpha \bullet B\beta],\ \epsilon,\ [A \to \alpha \bullet B\beta][B \to \bullet\, \gamma]) \in \Delta$,
  for all $(A \to \alpha B\beta), (B \to \gamma) \in P$;

- $([A \to \alpha \bullet a\beta],\ a,\ [A \to \alpha a \bullet \beta]) \in \Delta$,
  for all $(A \to \alpha a\beta) \in P$;

- $([A \to \alpha \bullet B\beta][B \to \gamma \bullet],\ \epsilon,\ [A \to \alpha B \bullet \beta]) \in \Delta$,
  for all $(A \to \alpha B\beta), (B \to \gamma) \in P$.

If we define f to map each stack to its top element (i.e. $f(\alpha q) = q$, for all $\alpha \in Q^*$ and $q \in Q$), then we obtain an approximation that is very close to our approximation from Section 3.; in fact, the two approximations are identical if all nonterminals belong to a single N_1 such that $recursive(N_1) = self$. This becomes even more clear if the approximation from Section 3. is expressed in a form inspired by recursive transition networks, as in (Nederhof, 2000).

The approximation above also concurs with the simplified approximation from (Grimley Evans, 1997) (omitting conditions 7 and 8 therein).

The same mapping f, but in combination with a different parsing strategy, viz. LR parsing, underlies an approximation in (Baker, 1981). By generalizing f to yield the top-most t elements of the stack (or the entire stack if it is less than t elements high), given some fixed parameter t, the approximation in (Bermudez and Schimpf, 1990) is obtained, also in combination with LR parsing.

Another approximation based on LR parsing is given in (Pereira and Wright, 1997), and is obtained by defining f to map each stack to another stack in which no stack symbol occurs more than once. Given a stack α, the stack $f(\alpha)$ is reached by iteratively replacing substacks of the form $q\beta q$ by q; in case there are several such substacks, one chooses one among them in a canonical way to ensure f is uniquely defined.

The main purpose of this paper is to clarify what happens during regular approximation of context-free languages. Our grammar-based method represented by Figure 1.5 can be seen as the simplest approach to remove self-embedding leaving other parts of the grammar unaffected, and as we have shown, removing self-embedding is sufficient for obtaining a regular language.

Given its simplicity, it is not surprising that more sophisticated methods, such as those based on the LR parsing strategy, produce strictly more precise approximations, i.e. regular languages that are smaller, in terms of language inclusion $\subseteq$. However, our experiments have shown that such sophistication sometimes deteriorates rather than improves practical usefulness.

For example, the method from (Baker, 1981) in many cases yields the same regular language as our method, yet the intermediate results in the approximation are much larger, which can be argued theoretically but is also confirmed by experiments reported in (Nederhof, 2000). The "sophistication" of LR parsing is here merely a source of needless inefficiency.

The high costs involved in applying the LR parsing strategy are even more apparent in the case of the method from (Pereira and Wright, 1997): First, the construction of an LR automaton, of which the size is exponential in the size of the grammar, may be a very expensive task in practice (Nederhof and Satta, 1996). This is however only a fraction of the effort needed for considering all stacks in which stack symbols occur at most once, i.e. the stacks obtained by applying f, which is in turn exponential in the size of the LR automaton.[3]

By contrast, the complexity of our approximation algorithm (Figure 1.5) is polynomial. The only exponential behaviour may come from the subsequent construction of the finite automaton (Figure 1.3), when the grammar is descended in all ways, a source of exponential behaviour which is also part of the methods based on LR parsing. There is a representation of the automata

however that also avoids this exponential behaviour (Nederhof, 2000; Mohri and Pereira, 1998).

That the grammar-based method in general produces less precise approximations may seem a weakness: *ad hoc* refinements such as those discussed in Section 5. may be needed to increase precision. For example, consider the grammar in Section 9 of (Church and Patil, 1982) and the 4-line grammar of noun phrases from (Pereira and Wright, 1997). Neither of these grammars are strongly regular, yet they generate regular languages. For the first grammar, the method from (Pereira and Wright, 1997) and the grammar-based method give the same result. But for the second grammar, the two methods give the same result only provided the second refinement from Section 5. is incorporated into our grammar-based method, with $j = 1$.

Our viewpoint is however that the methods relying on LR parsing also incorporate ad hoc mechanisms of obtaining approximations that are more precise than what is minimally needed to obtain a regular language, and that these are outside of the control of the user.

A case in point is the grammar of palindromes. In Section 5. we demonstrated how precision for our method can be improved in a controlled manner. However, the method from (Pereira and Wright, 1997) forces a result upon us which is given by $\epsilon \cup (a\{a,b\}^*a \ - \ a(ba)^*) \cup (b\{a,b\}^*b \ - \ b(ab)^*)$; in other words, just the left-most and right-most symbols are matched, but alternating series of a's and b's are excluded. This strange approximation is reached due to some intricate aspect of the structure of the LR automaton, and there is no reason to consider it as more natural or desirable than any other approximation, and although this method allows additional refinement as well, as shown by (Rood, 1996), the nature of such refinement may be that even more of the same kind of idiosyncrasies is introduced.

We now consider the method of approximation from (Grimley Evans, 1997), which is rephrased as follows. For each rule $A \rightarrow X_1 \cdots X_m$ we make a finite automaton with states $q_0, \ldots, q_m$ and transitions (q_{i-1}, X_i, q_i), where $1 \leq i \leq m$. These automata are then joined: Each transition (q_{i-1}, B, q_i), where B is a nonterminal, is replaced by a set of ϵ-transitions from q_{i-1} to the "left-most" states for rules with left-hand side B, and conversely, a set of ϵ-transitions from the "right-most" states for those rules to q_i.

This essentially replaces recursion by ϵ-transitions, which leads to a crude approximation (which is identical to the method above based on the top-down parsing strategy). An additional mechanism is now introduced that ensures that the list of visits to the states $q_0, \ldots, q_m$ belonging to a certain rule satisfies some reasonable criteria: a visit to q_i, with $0 \leq i < m$, should be followed by a visit to q_{i+1} or q_0. The latter option amounts to a nested incarnation of the rule. Similarly there is a condition for what should precede a visit to q_i, with $0 < i \leq m$. Since only pairs of consecutive visits to states from the set

$\{q_0, \ldots, q_m\}$ are considered, finite-state techniques suffice to implement such conditions.

The emphasis on treating rules individually has the consequence that the order of terminals in a string can become scrambled even when the approximation is still exact with respect to the *number* of occurrences of terminals. As in the case of the method from (Pereira and Wright, 1997), the resulting approximations are often surprising. E.g. for the palindrome grammar, the language we obtain is $\epsilon \cup a^2a^* \cup b^2b^* \cup (\Sigma^*a\Sigma^*b\Sigma^* \cup \Sigma^*b\Sigma^*a\Sigma^*)^2$, where $\Sigma = \{a, b\}$.

As reported in (Grimley Evans, 1997) and confirmed by the empirical data in (Nederhof, 2000), the resulting finite automata may be quite large, even for small grammars. The explanation is that conceptually a state of the finite automaton indicates for each grammar rule individually how far recognition of the right-hand side has progressed, which leads to exponential behaviour in the size of the grammar. It seems however that the method always results in approximations that are more precise than our grammar-based method without the refinements from Section 5..

Another way of obtaining a regular approximation of a grammar is to retain only the information about allowable pairs (or triples, etc.) of adjacent parts of speech (cf. bigrams, trigrams, etc.). This simple approach is proposed in (Herz and Rimon, 1991; Rimon and Herz, 1991), and is reported to be effective for the purpose of word tagging in Hebrew. (For extension to probabilistic formalisms, see (Stolcke and Segal, 1994).)

7. CONCLUSIONS

Comparing the respective superset approximations discussed above, we see that an important distinguishing property of our grammar-based method is that the structure of the context-free grammar is retained as long as possible as much as possible. This has two advantages. First, the remnants of the original structure present in the transformed grammar can be incorporated into the construction of the automaton, in such a way that the automaton (a finite transducer) produces output that can be used to build parse trees according to the original grammar. This has been shown in (Nederhof, 1998).

Secondly, the approximation process itself can be monitored: The author of a grammar can still see the structure of the old grammar in the new strongly regular grammar, and can in this way observe what kind of consequences the approximation has on the generated language.

Further comparison of different methods of approximation is provided in (Nederhof, 2000), which concentrates on two questions. First, what happens when a context-free grammar grows in size? What is then the increase of the size of the obtained minimal deterministic automaton? Second, how “precise”

are the approximations? That is, how much larger than the original context-free language is the language obtained by a superset approximation, and how much smaller is the language obtained by a subset approximation? An attempt is made to answer these questions by making measurements on a practical grammar for German and a corpus of spoken language.

Acknowledgments

The main part of this research was 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). Further support was obtained from the German Research Foundation (DFG), under grant Be1953/1-1, and from AT&T Labs.

Notes

1. We use an abbreviated notation for sets of context-free rules. Vertical lines separate respective right-hand sides of rules with identical left-hand sides.

2. The method from (Pereira and Wright, 1997) does a little better: of the strings of odd length it excludes those of length 1; yet it does allow all strings of length 3, 5, ... The alternative method from (Grimley Evans, 1997) excludes a, b, aba and bab, but allows all other strings of odd length. The methods are compared more closely in Section 6..

3. A suggestive example of the difficulty of applying the approximation from (Pereira and Wright, 1997) is provided by a grammar describing part of the Java programming language. Although two independent implementations we made have crashed before finishing the task, we can make a reasonable estimate of the costs that would be involved in constructing the (nondeterministic) finite automaton. The grammar has 28 nonterminals and 57 rules. The LR(0) characteristic machine has 120 states and 490 transitions. The number of stacks without multiple occurrences of states can be estimated as follows. We order the states as $q_0, \cdots, q_{119}$, and assume that from each q_k, with $0 \leq k \leq 115$, there are transitions to q_{k+1}, q_{k+2}, q_{k+3} and q_{k+4}; note that $4 * 116 = 464$, which is not far from 490, and therefore this scenario may be similar to the actual situation. One stack that can be constructed is $q_0 q_4 q_8 \cdots q_{116}$, but at least 29 times, another choice can be made which of 4 elements to push, starting from stack q_0, which results in different stacks. Thus there may be more than $4^{29} \approx 2.9 * 10^{17}$ different stacks and as many states in the finite automaton!

References

Baker, T. (1981). Extending lookahead for LR parsers. *Journal of Computer and System Sciences*, 22:243–259.

Bermudez, M. and Schimpf, K. (1990). Practical arbitrary lookahead LR parsing. *Journal of Computer and System Sciences*, 41:230–250.

Chomsky, N. (1959a). A note on phrase structure grammars. *Information and Control*, 2:393–395.

Chomsky, N. (1959b). On certain formal properties of grammars. *Information and Control*, 2:137–167.

Church, K. and Patil, R. (1982). Coping with syntactic ambiguity or how to put the block in the box on the table. *American Journal of Computational Linguistics*, 8:139–149.

Grimley Evans, E. (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.

Harrison, M. (1978). *Introduction to Formal Language Theory*. Addison-Wesley.

Herz, J. and Rimon, M. (1991). Local syntactic constraints. In *Proc. of the Second International Workshop on Parsing Technologies*, pages 200–209, Cancun, Mexico.

Hopcroft, J. and Ullman, J. (1979). *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley.

Johnson, M. (1998). Finite-state approximation of constraint-based grammars using left-corner grammar transforms. In *36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics*, volume 1, pages 619–623, Montreal, Quebec, Canada.

Krauwer, S. and des Tombe, L. (1981). Transducers and grammars as theories of language. *Theoretical Linguistics*, 8:173–202.

Langendoen, D. (1975). Finite-state parsing of phrase-structure languages and the status of readjustment rules in grammar. *Linguistic Inquiry*, 6(4):533–554.

Langendoen, D. and Langsam, Y. (1987). On the design of finite transducers for parsing phrase-structure languages. In Manaster-Ramer, A., editor, *Mathematics of Language*, pages 191–235. John Benjamins Publishing Company, Amsterdam.

Meyer, A. and Fischer, M. (1971). Economy of description by automata, grammars, and formal systems. In *IEEE Conference Record of the 12th Annual Symposium on Switching and Automata Theory*, pages 188–191.

Mohri, M. and Pereira, F. (1998). Dynamic compilation of weighted context-free grammars. In *36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics*, volume 2, pages 891–897, Montreal, Quebec, Canada.

Nederhof, M.J. (1994a). *Linguistic Parsing and Program Transformations*. PhD thesis, University of Nijmegen.

Nederhof, M.-J. (1994b). An optimal tabular parsing algorithm. In *32nd Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, pages 117–124, Las Cruces, New Mexico, USA.

Nederhof, M.-J. (1998). Context-free parsing through regular approximation. In *Proceedings of the International Workshop on Finite State Methods in Natural Language Processing*, pages 13–24, Ankara, Turkey.

Nederhof, M.-J. (2000). Practical experiments with regular approximation of context-free languages. *Computational Linguistics*, 26(1). In press.

Nederhof, M.-J. and Satta, G. (1996). Efficient tabular LR parsing. In *34th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, pages 239–246, Santa Cruz, California, USA.

Pereira, F. and Wright, R. (1997). Finite-state approximation of phrase-structure grammars. In Roche, E. and Schabes, Y., editors, *Finite-State Language Processing*, pages 149–173. MIT Press.

Pulman, S. (1986). Grammars, parsers, and memory limitations. *Language and Cognitive Processes*, 1(3):197–225.

Rimon, M. and Herz, J. (1991). The recognition capacity of local syntactic constraints. In *Fifth Conference of the European Chapter of the Association for Computational Linguistics, Proceedings of the Conference*, pages 155–160, Berlin, Germany.

Rood, C. (1996). Efficient finite-state approximation of context free grammars. In Kornai, A., editor, *Extended Finite State Models of Language*, Proceedings of the ECAI'96 workshop, pages 58–64, Budapest University of Economic Sciences, Hungary.

Rosenkrantz, D. and Lewis II, P. (1970). Deterministic left corner parsing. In *IEEE Conference Record of the 11th Annual Symposium on Switching and Automata Theory*, pages 139–152.

Sippu, S. and Soisalon-Soininen, E. (1990). *Parsing Theory, Vol. II: LR(k) and LL(k) Parsing*, volume 20 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag.

Stearns, R. (1967). A regularity test for pushdown machines. *Information and Control*, 11:323–340.

Stolcke, A. and Segal, J. (1994). Precise N-gram probabilities from stochastic context-free grammars. In *32nd Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, pages 74–79, Las Cruces, New Mexico, USA.

Ullian, J. (1967). Partial algorithm problems for context free languages. *Information and Control*, 11:80–101.

Valiant, L. (1975). Regularity and related problems for deterministic pushdown automata. *Journal of the ACM*, 22(1):1–10.