REGULAR APPROXIMATIONS OF CFLS: A GRAMMATICAL VIEW

Mark-Jan Nederhof
Dept. of Humanities Computing, University of Groningen
P.O. Box 716, NL-9700 AS Groningen, The Netherlands
Email: markjan@let.rug.nl

Abstract

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

1 Introduction

In [Pereira and Wright, 1991] a method was presented that allows the construction of a finite automaton from a context-free grammar. The (regular) language accepted by the finite automaton includes the strings generated by the original context-free grammar, plus a set of additional strings. In this sense, the context-free grammar is *approximated* by the finite automaton.

As intermediate result, an LR automaton is constructed from the context-free grammar. LR automata, which are special cases of pushdown automata, are representations of context-free languages. If one eliminates the pushdown stack of an LR automaton save the top-most element, a finite automaton results. For obtaining a better approximation, this stage can be preceded by encoding into stack symbols some bounded amount of information concerning the stack lying underneath; this information is given by a shorter stack resulting from omitting parts between multiple occurrences of LR states. This process is called *unfolding*.

In the context of error handling for programming languages, the approach from [Pereira and Wright, 1991], without the concept of unfolding, has been conceived independently by [Heckert, 1994].

Some aspects of this method have however not been clarified:

- Why are LR automata used, as opposed to any other kind of pushdown automaton? (See for example [Nederhof, 1994b] for a whole range of different kinds of pushdown automaton.)

- What happens to the language in the approximating process?

The second issue is partly a consequence of the first issue: the structure of a grammar and the structure of the corresponding LR automaton are two very different things, and how modifications to the automaton affect the language in terms of the grammar may be difficult to see.

In this paper, we will somewhat clarify these two issues. Concerning the first issue, we simply state that any kind of pushdown automaton constructed from a grammar allows a finite automaton to be derived that represents a regular approximation of the original language; in other words, LR automata do not have any special properties in this respect. However, in this communication, we will avoid the use of pushdown automata altogether, and perform the process of approximation on the level of the context-free grammar. This also helps to clarify the second issue above, viz. how to monitor the approximating process.

Our method 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.

$$S \rightarrow (S * S)$$
$$S \rightarrow (S + S)$$
$$S \rightarrow 0$$

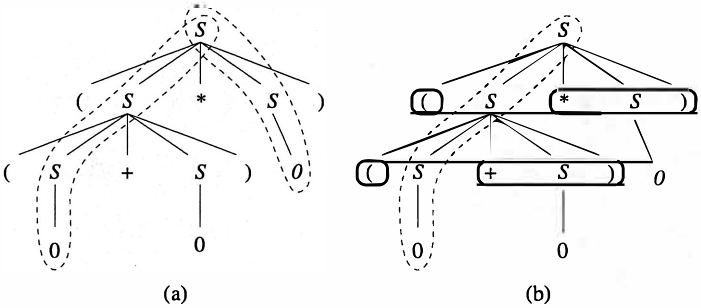


Figure 1: (a) two spines in a parse tree, (b) the grammar symbols to the left and right of a spine immediately dominated by nodes on the spine.

The structure of the 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.

An algorithm to transform a grammar such that this condition is satisfied is given in Section 4. As Section 5 shows, some aspects of our method are undecidable. A refinement of the approach for obtaining more precise approximations is presented in Section 6. Section 7 compares the new method to existing methods.

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 terminal and nonterminal symbols, respectively, $S \in N$ is the start symbol, and P is a finite set of rules. Each rule has the form $A \rightarrow \alpha$ with $A \in N$ and $\alpha \in V^*$, where V denotes $N \cup \Sigma$. The relation $\rightarrow$ on $N \times V^*$ is extended to a relation on $V^* \times V^*$ as usual. The transitive and reflexive closure of $\rightarrow$ is denoted by $\rightarrow^*$.

The language *generated* by a context-free grammar is given by the set $\{w \in \Sigma^* \mid S \rightarrow^* w\}$. By definition, such a set is a *context-free* language. By *reduction* of a grammar we mean the elimination from P of all rules $A \rightarrow \gamma$ such that $S \rightarrow^* \alpha A \beta \rightarrow \alpha\gamma\beta \rightarrow^* w$ 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 $A \rightarrow B$ is called a *unit* rule. A grammar is called *cyclic* if $A \rightarrow^+ A$, 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 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.

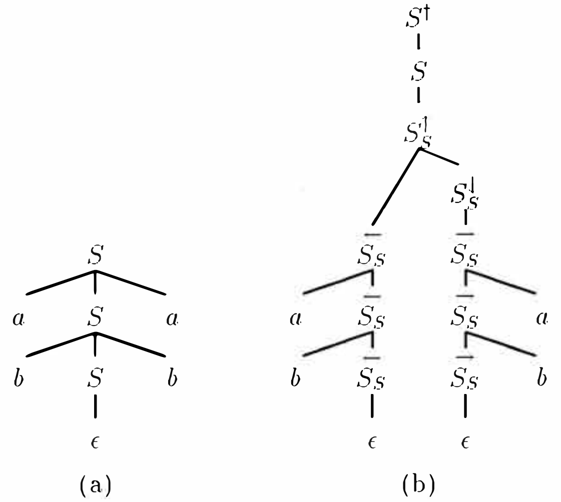
3 The Structure of Parse Trees

We define a *spine* in a parse tree to be a path that runs from the root down to some leaf. Figure 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 (b) gives an example: to the left of the spine we find the sequence "((" and to the right we find "$+S) * S)$". The way such pairs of sequences may relate to each other determines the strength of 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 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

$$\begin{array}{l} S \text{ — } a\ S\ a \\ S \text{ — } b\ S\ b \\ S \text{ — } \epsilon \end{array}$$



$$\begin{array}{l} S^\dagger \text{ — } S \\ \overline{S}_S \text{ — } a\ \overline{S}_S \\ \overline{S}_S \text{ — } b\ \overline{S}_S \\ \overline{S}_S \text{ — } \overline{S}_S\ a \\ \overline{S}_S \text{ — } \overline{S}_S\ b \\ \overline{S}_S \text{ — } \epsilon \\ \overline{S}_S \text{ — } \epsilon \\ S \text{ — } S^{\downarrow}_S \\ S^{\downarrow}_S \text{ — } \overline{S}_S\ S^{\downarrow}_S \\ S^{\downarrow}_S \text{ — } \overline{S}_S \end{array}$$

Figure 2: Parse trees for a palindrome: (a) original grammar, (b) transformed grammar (Section 4).

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 collection 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 $A \text{ —}^* \alpha A \beta$, 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 [Harrison, 1978]) 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:

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

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

$$\mathcal{N} = \{N_1, N_2, \ldots, N_m\}$$
$$N_1 \cup N_2 \cup \ldots \cup N_m = \overline{N}$$
$$N_i \cap N_j = \emptyset, \text{ for all } i, j \text{ such that } 1 \leq i < j \leq m$$
$$\exists i [A \in N_i \wedge B \in N_i] \quad \Leftrightarrow \quad \exists \alpha_1, \beta_1, \alpha_2, \beta_2 [A \text{ —}^* \alpha_1 B \beta_1 \wedge B \text{ —}^* \alpha_2 A \beta_2], \text{ for all } A, B \in \overline{N}$$

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

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

where

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

When *recursive*$(N_i) =$ *left*, N_i consist 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 *recursive*$(N_i) =$ *cyclic*, it is *only* such unit rules that take part in the recursion.

That *recursive*$(N_i) =$ *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 3 presents an algorithm for creating a finite automaton which 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 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 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.

4 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 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 5. After this approximation algorithm, the grammar will be strongly regular. We will explain the transformation by means of two examples.

The first example deals with the special case that each nonterminal can lead to at most one recursive call of itself.[1] Consider the grammar of palindromes in the left half of Figure 2. The approximation algorithm leads to the grammar in the right half. Figure 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 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

[1] This is the case for *linear* context-free grammars [Hopcroft and Ullman, 1979].

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

procedure $make\_fa(q_0, \alpha, q_1)$:
  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 $q = fresh\_state$;
         $make\_fa(q_0, X, q)$;
         $make\_fa(q, \beta, q_1)$
  else let $A = \alpha$;   (* $\alpha$ 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 $(C — X_1 \ldots X_m) \in P$ 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 $(C — DX_1 \ldots X_m) \in P$ 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 $(A — \beta) \in P$ do $make\_fa(q_0, \beta, q_1)$ 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 3: Transformation from a strongly regular grammar $G = (\Sigma, N, P, S)$ into an equivalent finite automaton $\mathcal{F} = (K, \Sigma, \Delta, s, F)$.

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.

The effect of our transformation on the structure of the parse tree is suggested in Figure 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.

For example, consider the following grammar for mathematical expressions:

$$\begin{array}{lcl} S & — & A * B \\ A & — & (A + B) \mid a \\ B & — & [A] \mid b \end{array}$$

We have $\overline{N} = \{A, B\}$, $\mathcal{N} = \{N_1\}$, $N_1 = \{A, B\}$. Note that $recursive(N_1) = self$. After applying the approximation algorithm to N_1 we obtain:

$$\begin{array}{lcl} S & \rightarrow & Aa \\ A & \rightarrow & SB \\ A & \rightarrow & Bb \\ B & \rightarrow & Bc \\ B & \rightarrow & d \end{array} \qquad \begin{array}{l} \overline{N} = \{S, A, B\} \\ \mathcal{N} = \{N_1, N_2\} \\ N_1 = \{S, A\} \\ N_2 = \{B\} \end{array}$$

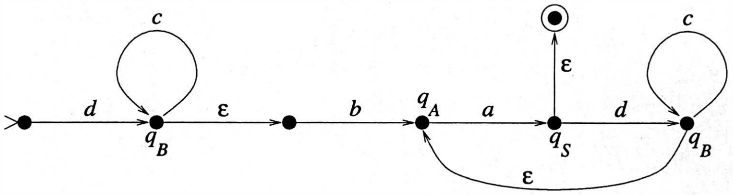


Figure 4: Application of the code from Figure 3 on a small grammar.

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. If $S \in N_i$, then augment the grammar with new nonterminal $S^{\dagger}$ and rule $S^{\dagger} \rightarrow S$ and choose $S^{\dagger}$ to be the new start symbol of the grammar.
2. Add the following nonterminals to N: $A^{\uparrow}_{B}$, $A^{\downarrow}_{B}$, $\overleftarrow{A_B}$ and $\overrightarrow{A_B}$ for all $A, B \in N_i$.
3. Add the following rules to P, for all $A, B, C, D, E \in N_i$:
    - $\overleftarrow{A_B} \rightarrow X_1 \ldots X_m \overleftarrow{C_B}$, for all $(A \rightarrow X_1 \ldots X_m C\beta) \in P$, with $X_1, \ldots, X_m \notin N_i$;
    - $\overrightarrow{A_B} \rightarrow \overrightarrow{C_B} X_1 \ldots X_m$, for all $(A \rightarrow \alpha C X_1 \ldots X_m) \in P$, with $X_1, \ldots, X_m \notin N_i$;
    - $\overleftarrow{A_A} \rightarrow \epsilon$;
    - $\overrightarrow{A_A} \rightarrow \epsilon$;
    - $A \rightarrow A^{\uparrow}_{A}$;
    - $A^{\uparrow}_{B} \rightarrow \overleftarrow{A_C} X_1 \ldots X_m C^{\downarrow}_{B}$, for all $(C \rightarrow X_1 \ldots X_m) \in P$, with $X_1, \ldots, X_m \notin N_i$;
    - $A^{\downarrow}_{B} \rightarrow \overrightarrow{C_A} X_1 \ldots X_m E^{\uparrow}_{B}$, for all $(D \rightarrow \alpha C X_1 \ldots X_m E\beta) \in P$, with $X_1, \ldots, X_m \notin N_i$;
    - $A^{\downarrow}_{B} \rightarrow \overrightarrow{B_A}$.
4. Remove from P the old rules of the form $A \rightarrow \alpha$, where $A \in N_i$.
5. Reduce the grammar.

Figure 5: Approximation by transforming the grammar, given a set N_i.

$$\begin{array}{lcl} S & \rightarrow & A * B \\ \overleftarrow{A_A} & \rightarrow & (\ \overleftarrow{A_A} \\ \overleftarrow{B_A} & \rightarrow & [\ \overrightarrow{A_A} \\ \overrightarrow{A_A} & \rightarrow & \overleftarrow{B_A} \ ) \\ \overrightarrow{A_B} & \rightarrow & \overleftarrow{B_B} \ ) \\ \overrightarrow{B_A} & \rightarrow & \overleftarrow{A_A} \ ] \\ \overrightarrow{B_B} & \rightarrow & \overleftarrow{A_B} \ ] \end{array} \qquad \begin{array}{lcl} \overleftarrow{A_A} & \rightarrow & \epsilon \\ \overleftarrow{B_B} & \rightarrow & \epsilon \\ \overrightarrow{A_A} & \rightarrow & \epsilon \\ \overrightarrow{B_B} & \rightarrow & \epsilon \\ A & \rightarrow & A^{\uparrow}_{A} \\ B & \rightarrow & B^{\uparrow}_{B} \end{array} \qquad \begin{array}{lcl} A^{\uparrow}_{A} & \rightarrow & \overleftarrow{A_A} \ a \ A^{\downarrow}_{A} \\ B^{\uparrow}_{A} & \rightarrow & \overleftarrow{B_A} \ a \ A^{\downarrow}_{A} \\ B^{\uparrow}_{A} & \rightarrow & \overleftarrow{B_B} \ b \ B^{\downarrow}_{A} \\ B^{\uparrow}_{B} & \rightarrow & \overleftarrow{B_A} \ a \ A^{\downarrow}_{B} \\ B^{\uparrow}_{B} & \rightarrow & \overleftarrow{B_B} \ b \ B^{\downarrow}_{B} \end{array} \qquad \begin{array}{lcl} A^{\downarrow}_{A} & \rightarrow & \overrightarrow{A_A} + B^{\uparrow}_{A} \\ B^{\downarrow}_{A} & \rightarrow & \overrightarrow{A_B} + B^{\uparrow}_{A} \\ A^{\downarrow}_{B} & \rightarrow & \overrightarrow{A_A} + B^{\uparrow}_{B} \\ B^{\downarrow}_{B} & \rightarrow & \overrightarrow{A_B} + B^{\uparrow}_{B} \\ A^{\downarrow}_{A} & \rightarrow & \overrightarrow{A_A} \\ B^{\downarrow}_{A} & \rightarrow & \overrightarrow{A_B} \\ A^{\downarrow}_{B} & \rightarrow & \overrightarrow{B_A} \\ B^{\downarrow}_{B} & \rightarrow & \overrightarrow{B_B} \end{array}$$

If we compare this example to the general picture in Figure 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 at the bottom. A similar fact holds for nonterminals such as $\overrightarrow{B_A}$. Nonterminals such as $B^{\uparrow}_{A}$ and

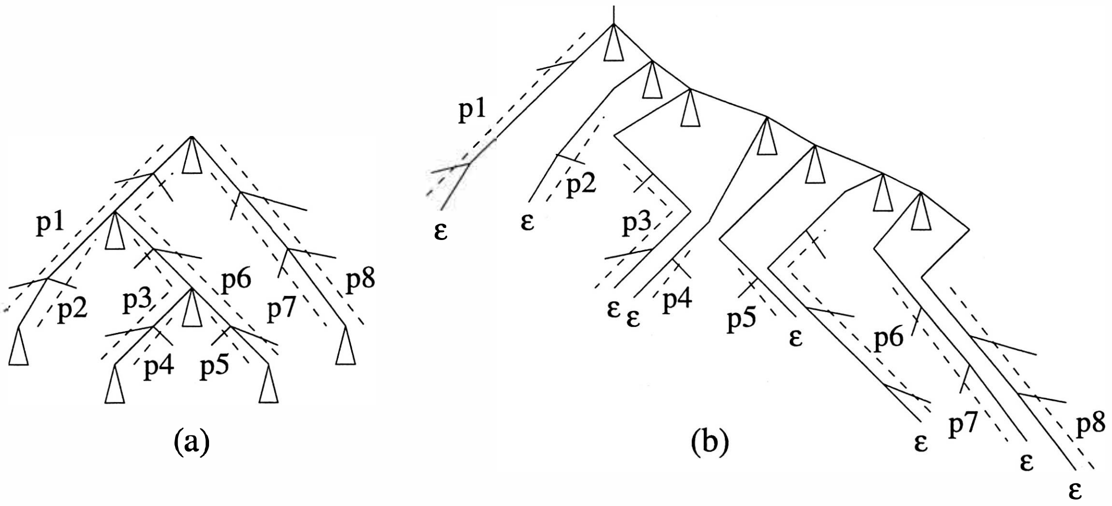


Figure 6: The general effect of the transformation on the structure of parse trees.

$B^{\downarrow}_{A}$ 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 $p8$.

5 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:

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

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

However, in some cases a grammar which is not strongly regular and generates a regular language may be transformed into one which generates a strictly larger language. This pertains to a claim made by [Pereira and Wright, 1991] that their method is always language-preserving when the original grammar already generates a regular language, which was refuted by a revised paper by the same authors.[2] It turns out that transforming a context-free grammar generating a regular language into a finite automaton accepting the same language is an unsolvable problem [Ullian, 1967], and consequently, the method from [Pereira and Wright, 1991] cannot satisfy this property, nor can our new method.

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.[3]

[2] Posted on the Computation and Language E-Print Archive, as number 9603002.

[3] The method from [Pereira and Wright, 1991] 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 two methods are compared more closely in Section 7.

If we represent context-free languages by means of pushdown automata, 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 corresponding finite automata is quite complex: The (deterministic) finite automata may require a number of states that is a double exponential function in the size of the original deterministic pushdown automata [Meyer and Fischer, 1971]; [Valiant, 1975] has shown that the *upper* bound to the number of states is also a double exponential function.

For arbitrary context-free grammars that describe 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].[4]

6 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 $recursive(N_i) = self$). One can however retain a finite amount of self-embedding by unfolding j levels of applications of nonterminals from N_i before the approximation algorithm is applied. This is done in such a way that in those j levels no recursion takes place and precision thus remains unaffected.

More precisely, for each nonterminal $A \in N_i$ we introduce j fresh nonterminals $A[1], \ldots, A[j]$, and we change the rules of the (augmented) grammar as follows.

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

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

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

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

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

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

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 imagine 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 the grammar is not cyclic.[5] A more complicated way of achieving such approximations, using refinement of the method from [Pereira and Wright, 1991], was discussed in [Rood, 1996].

This language-preserving transformation 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 $A \in N_i$. 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: $\{k_1, \ldots, k_p\} = \{k \mid 1 \leq k \leq m \wedge X_k \in N_i\}$

[4] This is equivalent to stating that transformation of such a context-free grammar into a finite automaton accepting the same language is an unsolvable problem, which we mentioned before.

[5] This problem of cyclic grammars can be easily overcome, but this is beyond the scope of the present paper.

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

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

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 that the distance to the nearest ancestor node not in N_i is more than j.

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

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

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

7 Comparison

The main purpose of this paper is to clarify what happens during the process of finding regular approximations of context-free languages. Our grammar-based method represented by Figure 5 can be seen as the simplest approach to remove self-embedding, 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 the LR-based method from [Pereira and Wright, 1991], produce strictly more precise approximations, i.e. regular languages that are smaller, in terms of language inclusion $\subseteq$. However, our empirical experiments have shown that such sophistication sometimes deteriorates rather than improves practical usefulness of the method.

For example, the appendix of [Pereira and Wright, 1991] contains a small grammar which is already strongly regular. The authors of that paper report they obtain a finite automaton with 2615 states and 4096 transitions before determinization and minimization. For our method however, no approximation at all is needed, and construction of the finite automaton by means of the algorithm in Figure 3 produces a finite automaton with only 957 states and 2751 transitions. After determinization and minimization of course the same automaton results. This example suggests that much of the "sophistication" of the LR-based method entails wasted computational overhead.

We have also considered the grammar in Section 9 of [Church and Patil, 1982] and the 4-line grammar of noun phrases from [Pereira and Wright, 1991]. Neither of the grammars are strongly regular, yet they generate regular languages. For the first grammar, the LR-based and the grammar-based methods give the same results. For the second grammar, the two methods give the same result only provided the second refinement from Section 6 is incorporated into our grammar-based method, taking $j = 1$.

That the grammar-based method in general produces less precise approximations may seem a weakness: *ad hoc* refinements such as those discussed in Section 6 may be needed to increase precision. Our viewpoint is that the LR-based method also incorporates ad hoc mechanisms of obtaining approximations that are more precise than what is minimally needed to obtain a regular language, but that these are however outside of the control of the user.

A case in point is the grammar of palindromes. In Section 6 we demonstrated how precision for our method can be improved in a controlled manner. However, the LR-based method forces a result upon us which is given by $(a\{a,b\}^*a \Leftrightarrow a(ba)^*) \cup (b\{a,b\}^*b \Leftrightarrow b(ab)^*)$; 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 the LR-based 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.

In addition, we point out that construction of an LR automaton, of which the size is exponential in the size of the grammar, may be a prohibitively expensive task in practice [Nederhof and Satta, 1996]. This is however only a fraction of the effort needed for the unfolding step of the LR-based method, which is in turn exponential in the size of the LR automaton. This became apparent when we tried to apply two independently developed implementations of the LR-based method on a subgrammar for the Java programming language. Both implementations crashed due to excessive use of memory, which is in keeping with our estimates that the number of LR stacks that need to be considered in this case may be astronomical. By contrast, the complexity of our approximation algorithm (Figure 5) is polynomial. The only exponential behaviour may come from the subsequent construction of the finite automaton (Figure 3), when the grammar is descended in all ways, a source of exponential behaviour which is also part of the LR-based method. Our implementation produced a nondeterministic finite automaton from the Java subgrammar within 12 minutes.

Recently, [Grimley Evans, 1997] has proposed a third approach, which I rephrase as follows. We consider a context-free grammar as a recursive transition network [Woods, 1970]: 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), $1 \leq i \leq m$. 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. 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, $0 \leq i < m$, should be followed by one 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, $0 < i \leq m$. 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.

If we compare that approach to our grammar-based method combined with e.g. the second refinement from Section 6, we can roughly say that the difference is that while our approach provides exact approximations for trees up to a certain height, the approach by [Grimley Evans, 1997] provides an exact approximation when no nested incarnations of individual rules occur. This emphasis on treating rules individually has the consequence that the order of terminals in a string can become mixed-up even when the approximation is still exact with respect to the *number* of occurrences of terminals; in effect, distinct rules interact in a way that is not consistent with the recursive structure of the original grammar.

It seems that the approach by [Grimley Evans, 1997] always results in approximations that are more precise than our approach without the refinements from Section 6.

An important difference of our grammar-based method with both the LR-based method and the one from [Grimley Evans, 1997] 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, including the names of the nonterminals, can be incorporated into the construction of the finite automaton, in such a way that the automaton produces output which can be used to build parse trees according to the original grammar. The algorithm from Figure 3 can be easily extended to construct such a finite *transducer*, using ideas from [Langendoen, 1975; Krauwer and des Tombe, 1981; Langendoen and Langsam, 1990].

Secondly, the approximation process itself can be monitored easily: 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.

Regarding other related work, I hesitate to mention [Langendoen and Langsam, 1987], which seems to be concerned more with psycho-linguistic arguments than with any level of mathematical accuracy. As the LR-based method, this approach seems to be based on pushdown automata, which here implement a particularly incorrect variant of left-corner parsing, allowing ungrammatical sentences to be accepted, which seems to be unintentional. The manner in which finiteness of the automaton is achieved is by requiring a collapse of a certain group of elements already in the parsing stack (if such a group of elements is at all present) into a smaller combination of stack elements, upon finding that some stack element has more than 2 occurrences. This collapse represents a forced attachment of previously unconnected subtrees in the parse tree. The effect is obviously a reduction of the language. However, due to the incorrectness of the variant of left-corner parsing, the resulting language may also contain sentences *not* in the original language. This makes comparison with other methods difficult, if not pointless.

The intended purpose of the above work may however be to reduce, as opposed to extend, a context-free language to become an (infinite) regular language.

Related to this is [Krauwer and des Tombe, 1981]: two kinds of pushdown automaton are presented that implement top-down and left-corner parsing, respectively. The regular approximation results simply by putting an upper-limit on the size of the stack, thus restricting the language. Unpublished work by Mark Johnson[6] solves the task in the same vein. The idea is extended to feature grammars in [Black, 1989].

Some theoretical limitations of these ideas 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].

Another way to obtain a finite-state 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 by [Herz and Rimon, 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].)

Acknowledgements

Special thanks go to René Deist for collaboration during my stay at the University of Bochum. I thankfully acknowledge valuable discussions with Gertjan van Noord, Mark Johnson, Eberhard Bertsch, Anton Nijholt and Rieks op den Akker.

This research was carried out within the framework of the Priority Programme Language and Speech Technology (TST). 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.

References

[Black, 1989] A.W. Black. Finite state machines from feature grammars. In *International Workshop on Parsing Technologies*, pages 277–285, Pittsburgh, 1989.

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

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

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

[Grimley Evans, 1997] E. Grimley Evans. Approximating context-free grammars with a finite-state calculus. In *35th Annual Meeting of the ACL*, pages 452–459, Madrid, Spain, July 1997.

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

[Heckert, 1994] E. Heckert. *Behandlung von Syntaxfehlern für LR-Sprachen ohne Korrekturversuche*. PhD thesis, Ruhr-Universität Bochum, 1994.

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

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

[6] *Left Corner Transforms and Finite State Approximations*, draft, May 1996.

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

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

[Langendoen and Langsam, 1990] D.T. Langendoen and Y. Langsam. A new method of representing constituent structures. *Annals New York Academy of Sciences*, 583:143–160, 1990.

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

[Meyer and Fischer, 1971] A.R. Meyer and M.J. Fischer. 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, 1971.

[Nederhof and Satta, 1996] M.J. Nederhof and G. Satta. Efficient tabular LR parsing. In *34th Annual Meeting of the ACL*, pages 239–246, Santa Cruz, California, USA, June 1996.

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

[Nederhof, 1994b] M.J. Nederhof. An optimal tabular parsing algorithm. In *32nd Annual Meeting of the ACL*, pages 117–124, Las Cruces, New Mexico, USA, June 1994.

[Pereira and Wright, 1991] F.C.N. Pereira and R.N. Wright. Finite-state approximation of phrase structure grammars. In *29th Annual Meeting of the ACL*, pages 246–255, Berkeley, California, USA, June 1991.

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

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

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

[Stolcke and Segal, 1994] A. Stolcke and J. Segal. Precise n-gram probabilities from stochastic context-free grammars. In *32nd Annual Meeting of the ACL*, pages 74–79, Las Cruces, New Mexico, USA, June 1994.

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

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

[Woods, 1970] W.A. Woods. Transition network grammars for natural language analysis. *Communications of the ACM*, 13(10):591–606, October 1970.