Chapter 9

Regular Approximation of Context-Free Grammars through Transformation

Mehryar Mohri
AT&T Labs — Research
mohri@research.att.com

Mark-Jan Nederhof
AT&T Labs — Research
nederhof@research.att.com

Abstract We present an algorithm for approximating context-free languages with regular languages. The algorithm is based on a simple transformation that applies to any context-free grammar and guarantees that the result can be compiled into a finite automaton. The resulting grammar contains at most one new nonterminal for any nonterminal symbol of the input grammar. The result thus remains readable and if necessary modifiable. We extend the approximation algorithm to the case of weighted context-free grammars. We also report experiments with several grammars showing that the size of the minimal deterministic automata accepting the resulting approximations is of practical use for applications such as speech recognition.

9.1 Introduction

Despite the availability of extensive literature on the topic of efficient context-free parsing, for large and very ambiguous grammars, context-free parsing poses a serious problem in many practical applications such as real-time speech recognition. For most grammars used in those applications, rules are annotated with weights, and efficient processing of weights forms an additional challenge for the implementation of context-free parsers.

Much more attractive computational properties can be attributed to the use of finite (weighted) automata both in theory and in practice. This commu-

Jean-Claude Junqua and Gertjan van Noord (eds), Robustness in Language and Speech Technology. Kluwer Academic Publishers, 2000, pp. 251–261.

nication deals with ideas that allow us to make use of these advantageous properties by approximating context-free languages with regular languages. Such (weighted) approximations could play a crucial role in constructing robust speech recognition systems because they are computationally less demanding than context-free grammars and because in general they can give a more accurate model of the syntactic and semantic properties of natural languages than classical n-gram language models.

Several approximations have been described in the existing literature, among which are Pereira and Wright [1997], Grimley Evans [1997], and Johnson [1998]; an extensive bibliography of approximations can be found in Nederhof [2000]. None of these approximations however provides much insight into how the language is changed during the approximation process, and it is difficult or impossible to influence this process in order to fine-tune the approximating language to the application.

For example, Pereira and Wright [1997] proposes the construction of a specific kind of pushdown automaton from the grammar, which is subsequently approximated to be a finite automaton. Since here the structure of a pushdown automaton is so different from the structure of the context-free language from which it is constructed, it is close to impossible for the grammar writer to predict or influence the approximating language that will result based on his understanding of the grammar.

Our objective is to solve this problem by defining an approximation through a simple transformation of the original grammar. The grammar writer may inspect the transformed grammar, in which the structure of the original grammar is still visible, and change it in a way suitable for the application.

Our approximation algorithm applies to any context-free grammar and guarantees that the result can be compiled into a finite automaton. The resulting grammar contains at most one new nonterminal for any nonterminal symbol of the input grammar, and new rules are formed out of rules from the input grammar by means of a straightforward decomposition. The result thus remains readable and if necessary modifiable. We show that the algorithm can be extended to the case of weighted context-free grammars.

Experiments with several grammars show that the size of the minimal deterministic automata accepting the resulting approximations is of practical use for applications such as speech recognition.

9.2 Preliminaries

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

$$L(G) = \{w \in \Sigma^* : S \stackrel{*}{\rightarrow} w\}$$

and is called a *context-free language*. We denote the empty string by ϵ. We let $|G|$ denote the number of rules of a grammar G. We also define the size of G as the total number of occurrences in P of symbols from V, and denote it by $\|G\|$. 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 further consider *regular languages*, the class of languages accepted by finite automata. We assume that the reader is familiar with these concepts; for more details we refer to Hopcroft and Ullman [1979].

9.3 Algorithm

9.3.1 Analysis and Compilation

Context-free grammars may generate languages that are not regular. We describe a subclass of grammars, *strongly regular grammars*, that are guaranteed to generate regular languages. This class of grammars coincides with that of grammars without self-embedding [Chomsky 1959]. Furthermore, strongly regular grammars can be mapped into equivalent finite automata using an efficient algorithm.

We then present our approximation algorithm, which transforms any grammar into one that is strongly regular. Note that a mapping from an arbitrary CFG generating a regular language into a corresponding finite automaton cannot be realized by any algorithm [Ullian 1967]. Therefore, neither our approximation, nor any other, can guarantee that the language is preserved when the grammar already generates a regular language. However, this is guaranteed when the grammar is strongly regular.

Let $\mathcal{R}$ be the relation defined on the set of nonterminals N of G by:

$$A \,\mathcal{R}\, B \;\Leftrightarrow\; (\exists \alpha, \beta \in V^* : A \stackrel{*}{\rightarrow} \alpha B \beta) \wedge (\exists \alpha, \beta \in V^* : B \stackrel{*}{\rightarrow} \alpha A \beta)$$

It is not hard to show that $\mathcal{R}$ defines an equivalence relation. $\mathcal{R}$ partitions N into subsets called sets of *mutually recursive nonterminals*.

For convenience, we refer to the rules with left-hand side $A \in N$ as *the rules of* A, and more generally, to the rules with left-hand side $A \in M$, for some $M \subseteq N$, as *the rules of* M. *Strongly regular grammars* are grammars in which the rules of each set M of mutually recursive nonterminals are either all right-linear or all left-linear.[1]

1. Nonterminals that do not belong to M are considered as terminals here, for determining if a rule of M is right-linear or left-linear.

There exist efficient algorithms for computing a finite automaton accepting the language generated by a strongly regular grammar. The finite automaton can be constructed off-line, as shown in Nederhof [1997], but as proposed by Mohri and Pereira [1998] one may also construct an alternative, compact representation of the regular language, from which the finite automaton may be computed; this compact representation may however also be used on-line for processing of input. The steps of this construction and the processing of input are sketched as follows:

1. determine the sets of mutually recursive nonterminals. This can be done in linear time in the size of the input grammar G by computing the strongly connected components of the graph of the grammar.[2]
2. construct a finite-state machine $\mathcal{K}(M)$ for each set of mutually recursive nonterminals M following the classical construction of an automaton from a regular grammar [Aho and Ullman 1973]. $\mathcal{K}(M)$ is in effect a finite automaton for which the initial state (in the case of a right-linear set M) or the set of final states (in the case of a left-linear set) have been left unspecified. A finite automaton $\mathcal{N}(A)$ describing the language corresponding to a nonterminal $A \in M$ can be easily derived from $\mathcal{K}(M)$ by specifying the state corresponding to the non-terminal A as an initial or final state depending on the type of the set (right-linear or left-linear case). Thus, $\mathcal{K}(M)$ combined with the start symbol S of the grammar now give a compact representation of the approximating language.
3. for each input string w, we first obtain $\mathcal{N}(S)$ from the $\mathcal{K}(M)$ that satisfies $S \in M$, and this automaton is then expanded in a lazy way by substituting other automata $\mathcal{N}(A)$ for occurrences of A in $\mathcal{N}(S)$ that are encountered while processing w.

Thus, the states and transitions of the finite automaton for the complete language are constructed on demand, when needed for the recognition of an actual input string. Moreover, the construction of the compact representation is optimal in the sense that it requires work linear in the size of the input grammar, or formally $\mathcal{O}(\|G\|)$. The compact representation can be further optimized using ϵ-removal, determinization, and minimization algorithms, possibly in combination with the substitution of subautomata.

9.3.2 Transformation

We now describe a grammar transformation that creates strongly regular grammars. For each set of mutually recursive nonterminals M such that the corresponding rules are not all right-linear or not all left-linear with respect to the nonterminals of M, we apply a grammar transformation defined as follows:[3]

1. For each nonterminal $A \in M$, introduce a new nonterminal $A' \notin N$, and

2. The *graph* of a grammar has one node for each nonterminal and has an edge from node A to node B if nonterminal B appears on the right-hand side of a rule with left-hand side A.
3. Other rules of the input grammar are left unchanged.

add the following rule to the grammar:[4]

$$A' \rightarrow \epsilon$$

2. Consider each rule with left-hand side $A \in M$:

$$A \rightarrow \alpha_0\ B_1\ \alpha_1\ B_2\ \alpha_2\ \cdots\ B_m\ \alpha_m$$

with $m \geq 0$, $B_1, \ldots, B_m \in M$, $\alpha_0 \ldots \alpha_m \in (\Sigma \cup (N - M))^*$, and replace it by the following set of rules:

$$\begin{array}{rcl} A & \rightarrow & \alpha_0\ B_1 \\ B'_1 & \rightarrow & \alpha_1\ B_2 \\ B'_2 & \rightarrow & \alpha_2\ B_3 \\ & \ldots & \\ B'_{m-1} & \rightarrow & \alpha_{m-1}\ B_m \\ B'_m & \rightarrow & \alpha_m\ A' \end{array}$$

(In the case where $m = 0$, this set of rules merely contains $A \rightarrow \alpha_0\ A'$.) Since the rules of M are replaced by right-linear rules, the resulting grammar is strongly regular and can be compiled into a finite automaton as discussed above.

An attractive property of the grammar transformation is that it can be applied to large grammars: at most one new nonterminal is introduced for each nonterminal of the input grammar, and the size of the resulting grammar is at most twice that of the input grammar.

A further convenient property is that the transformed grammar can be used for parsing. The nonterminals of the form X and X' correspond to the beginning and end of recognition of strings generated by X in the original grammar. This fact can be used to compile the transformed grammar into a finite-state transducer that outputs bracketed strings equivalent to parse trees. The resulting parse trees retain much of the structure of the original grammar. See Nederhof [1998] for a related idea.

The language generated by the transformed grammar is a superset of that of the original grammar. Indeed, it is clear from the way a rule in the original grammar is split into several rules, that any string accepted by a series of derivations in the input grammar is also accepted in the resulting grammar.

Since the size of the result of the transformation is comparable to that of the input grammar and since the symbols that are newly introduced still admit a syntactic interpretation, the resulting grammar remains readable, and if necessary can be modified by the user. Finally, we shall see that in several experiments the transformation also leads to automata of reasonable size and therefore that it is of practical interest.

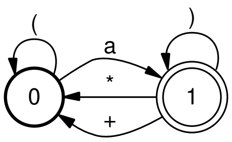
4. This can be refined by adding the rule only when A is directly reachable from another strongly connected component.

$$
\begin{array}{lcl}
E & \to & E + T \\
E & \to & T \\
T & \to & T * F \\
T & \to & F \\
F & \to & (\ E\ ) \\
F & \to & a
\end{array}
$$

(A)

$$
\begin{array}{lcl@{\qquad}lcl}
E' & \to & \epsilon & T & \to & T \\
T' & \to & \epsilon & T' & \to & *\ F \\
F' & \to & \epsilon & F' & \to & T' \\
E & \to & E & T & \to & F \\
E' & \to & +\ T & F' & \to & T' \\
T' & \to & E' & F & \to & (\ E \\
E & \to & T & E' & \to &)\ F' \\
T' & \to & E' & F & \to & a\ F'
\end{array}
$$

(B)



(C)

FIGURE 9.1: Regular approximation of a simple context-free grammar. (A) Grammar of arithmetic expressions. (B) Transformed grammar. (C) Finite automaton accepting the language generated by the transformed grammar.

Our approximation algorithm works in two stages: first the grammar is approximated using the transformation just defined. Then the resulting grammar is compiled into a finite automaton. As mentioned before, the compilation can be performed on demand, in which case the automaton is only constructed and expanded as far as it is required for the recognition of the input string, or the automaton can be completely expanded and optimized. Both options can be useful in practice, and the choice depends on time and space trade-offs. Note that as long as we do not substitute or optimize the subautomata, the total construction comprising the grammar transformation and the compilation is linear in the size of the grammar.

The grammar transformation is illustrated by figures 9.1-(a) and 9.1-(b). Figure 9.1-(a) is a simple grammar for well-formed arithmetic expressions. Figure 9.1-(b) shows the result after transformation. In the original grammar, the nonterminals E, T, and F can be interpreted as corresponding to an expression, a term, or a factor. In the transformed grammar, F for example corresponds to a suffix of an expression starting with a factor, and F' corresponds to a suffix of an expression just after the end of a factor.

The transformed grammar is strongly regular and can be compiled into the finite automaton of figure 9.1-(c).

Our grammar transformation bears some similarity to a transformation from Nederhof [1997] [equivalent to the RTN approximation from Nederhof 2000]. In fact, that approximation can be viewed as a more refined variant of our new transformation. A disadvantage of the old approximation method is however that it requires a quadratic number ($\mathcal{O}(|N|^2)$) of nonterminals in the resulting grammar, which may be unacceptable for some large grammars used in practice.

Nevertheless, in light of the approximation method just presented, we can give a simpler formulation of the older approximation as follows. For any set of mutually recursive nonterminals M, the grammar is transformed in the following steps.

1. Let M' be defined by: $A \in M'$ if and only if $A \in M$ and either $A = S$ or $\exists(B \to \alpha A \beta) \in P : B \notin M$.
2. Introduce two new nonterminals $A^B\!\uparrow$ and A^B for each $A \in M$ and $B \in M'$.
3. For each nonterminal $A \in M'$ add the following rule to the grammar:

$$A^A\!\uparrow \;\to\; \epsilon$$

4. Consider each rule with left-hand side $A \in M$:

$$A \;\to\; \alpha_0\; C_1\; \alpha_1\; C_2\; \alpha_2\; \cdots\; C_m\; \alpha_m$$

with $m \geq 0$, $C_1, \ldots, C_m \in M$, $\alpha_0 \ldots \alpha_m \in (\Sigma \cup (N - M))^*$, and replace it by the following set of rules, for each $C \in M'$:

$$\begin{array}{rcl} A^C & \to & \alpha_0\; B_1^C \\ B_1^C\!\uparrow & \to & \alpha_1\; B_2^C \\ B_2^C\!\uparrow & \to & \alpha_2\; B_3^C \\ & \ldots & \\ B_{m-1}^C\!\uparrow & \to & \alpha_{m-1}\; B_m^C \\ B_m^C\!\uparrow & \to & \alpha_m\; A^C\!\uparrow \end{array}$$

5. Replace each occurrence of $A \in M'$ in the old rules of the grammar by A^A.

The subset $M' \subseteq M$ represents the set of nonterminals in M that may be reached from parts of the grammar that are not involved in recursion in the set M; also the start symbol S can be in this set. Such nonterminals are at the root of a subtree of the parse tree constructed from nonterminals in M. These nonterminals are maintained in the superscripts of the new nonterminals, which is needed to ensure that the approximating grammar commits itself to a single nonterminal at the root of such a subtree.

Omitting the superscripts in the transformed grammar leads to the transformation presented before: A^B simplifies to A, and $A^B\!\uparrow$ to A', irrespective of B.

9.4 Weighted grammars

Grammars used in many applications such as those related to speech processing incorporate weights. These weights, which are often interpreted as probabilities, are used to rank different hypotheses for the purpose of disambiguation.

A *weighted context-free grammar* is a context-free grammar in which rules are additionally annotated with weights. The weight set $\mathbb{K}$ has in general the algebraic structure of a semiring $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$, with $\overline{0}$ as the identity element of $\oplus$ and $\overline{1}$ the identity element of $\otimes$. Weights from rules used in a single derivation are combined using the $\otimes$-product and the weights from alternative derivations (in the case of ambiguous grammars) are combined using the $\oplus$-sum. We will assume here that the $\oplus$-sum of the weights is well-defined and in $\mathbb{K}$ in all such cases.

The approximation method through transformation of context-free grammars can be extended to the weighted case in at least the following three cases:

- An unweighted grammar and a corpus are given. The grammar is transformed and an equivalent finite automaton is constructed from it. Weights are then assigned to its transitions by applying a learning algorithm based on the corpus.
- A weighted grammar is given. A corpus is constructed from that grammar and the weights are removed from the grammar. The rest of the process is identical to the previous case.
- A weighted grammar is given. The transformation is extended to assign weights to the output rules in such a way that the weight of a rule broken up into m rules by the transformation equals the $\otimes$-product of those m rules.

Here, we consider this last case and thus extend our transformation to weighted grammars as follows. For each set of mutually recursive nonterminals M such that the corresponding rules are not all right-linear and not all left-linear, the following transformation is applied:

1. For each nonterminal symbol $A \in M$, introduce a new nonterminal $A' \notin N$, and add the following rule with weight $\overline{1}$ to the grammar:

$$A' \;\rightarrow\; \epsilon \;\;/\overline{1}$$

2. Consider each rule with left-hand side $A \in M$ and weight $x \in \mathbb{K}$:

$$A \rightarrow \alpha_0\ B_1\ \alpha_1\ B_2\ \alpha_2\ \cdots\ B_m\ \alpha_m\ \ /x$$

with $m \geq 0$, $B_1, \ldots, B_m \in M$, $\alpha_0 \ldots \alpha_m \in (\Sigma \cup (N - M))^*$, and replace it by the following set of rules:

$$\begin{array}{rcl} A & \rightarrow & \alpha_0\ B_1\ \ /x_0 \\ B'_1 & \rightarrow & \alpha_1\ B_2\ \ /x_1 \\ B'_2 & \rightarrow & \alpha_2\ B_3\ \ /x_2 \\ & \cdots & \\ B'_{m-1} & \rightarrow & \alpha_{m-1}\ B_m\ \ /x_{m-1} \\ B'_m & \rightarrow & \alpha_m\ A'\ \ /x_m \end{array}$$

with $x_0, x_1, \ldots, x_m \in \mathbb{K}$, and $x_0 \otimes x_1 \otimes \cdots \otimes x_m = x$.

TABLE 9.1: Approximation experiments.

G	$\|G\|$	$\|G_{app}\|$	$\|A_{exp}\|$	$\|A\|$	transf. (s)	compil. (s)	opt. (s)
G_1	19	39	104	31	.14	.11	.40
G_2	47	82	354	95	.15	.12	.37
G_3	516	852	17121	3220	.24	.23	54
G_4	1434	2351	141304	1828	.35	.41	36
G_5	1641	4153	6924	7467	.59	.74	54
G_6	846	1992	55070	97	.38	.53	16

There exists at least one admissible choice for the values of $x_0, x_1, \ldots, x_m$ defined by: $x_0 = x$, and $x_1 = \cdots = x_m = \overline{1}$ in any semiring, but in general the weight x can be distributed over the new rules in many different ways.

Let D be a derivation in the original weighted grammar which derives string w with weight x. Then there will be a derivation D' deriving w with the same weight x in the transformed grammar. However, the transformed grammar may additionally admit derivations of strings w with weights for which no corresponding derivation can be found in original grammar.

As in the unweighted case, the resulting weighted grammar can be compiled into a weighted automaton [Mohri and Pereira 1998].

9.5 Experiments

We have a full implementation of the approximation algorithm presented in the previous section in the general case of weighted context-free grammars. The algorithm and the corresponding utilities have been incorporated in a set of general-purpose grammar tools, the GRM library [Mohri 2000].

Using that implementation, we carried out experiments with grammars of small to large sizes. Grammars G_1, G_2, G_3 and G_4 were used in various experiments by Nederhof [2000]. Our approximation results in these experiments exactly coincide with those obtained using the RTN approximation from Nederhof [2000]. Grammar G_5 was obtained from Carroll [1993], and grammar G_6 from Schoorl and Belder [1990].

Table 9.1 illustrates our approximation results for each of these grammars. The second column gives the number of rules of each grammar. The next column gives the number of rules of the transformed grammar. Column $|A_{exp}|$ gives the size (in terms of the number of transitions) of the expanded automaton A_{exp} recognizing the transformed grammar, as produced by our algorithm. The next column gives the size of the minimal deterministic automaton A equivalent to A_{exp}. The time required to apply the grammar transformation is indicated in column *transf.*, the time to compile that grammar into A_{exp} is given in column *compil.*, and the time needed to compute the minimal deterministic automaton A, by optimizing the subautomata in combination with expansion, is given in column *opt.*, using an SGI Origin 2000.

The results show that both the size of the transformed grammar and the size of the resulting automata are small enough for practical use for demanding applications such as real-time speech recognition. This contrasts with other existing approximation methods that were shown to produce automata of very large size even for small grammars of less than fifty rules [Nederhof 2000]. The approximation time including the two stages of transformation and compilation into a finite automaton is extremely fast and therefore does not present any practical obstacle.

We also applied our approximation algorithm to a weighted grammar of about $25,000$ rules used for translation at AT&T. The transformed grammar had about $36,000$ rules. The whole approximation process including the creation of a finite automaton accepting that grammar took about one minute using the same algorithm on the same machine.

9.6 Conclusion

We presented an algorithm for approximating context-free languages with regular languages. The algorithm was shown to be efficient. Experiments with several grammars showed that the size of the resulting automata is practical for use in many applications.

References

Aho, A. V. and Ullman, J. D.: 1973, *The Theory of Parsing, Translation and Compiling*, Prentice-Hall.

Carroll, J. A.: 1993, Practical unification-based parsing of natural language, *Technical Report No. 314*, University of Cambridge, Computer Laboratory, England. PhD thesis.

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

Grimley Evans, E.: 1997, Approximating context-free grammars with a finite-state calculus, *35th Annual Meeting of the ACL*, pp. 452–459.

Hopcroft, J. E. and Ullman, J. D.: 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, *36th Annual Meeting of the ACL and 17th International Conference on Computational Linguistics*, Vol. 1, pp. 619–623.

Mohri, M.: 2000, Weighted grammar tools: The GRM library, *in* J.-C. Junqua and G. van Noord (eds), *Robustness in Language and Speech Technology*, Kluwer Academic Publishers. This volume.

Mohri, M. and Pereira, F. C. N.: 1998, Dynamic compilation of weighted context-free grammars, *36th Annual Meeting of the ACL and 17th International Conference on Computational Linguistics*, Vol. 2, pp. 891–897.

Nederhof, M.-J.: 1997, Regular approximations of CFLs: A grammatical view, *International Workshop on Parsing Technologies*, Massachusetts Institute of Technology, pp. 159–170.

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

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

Pereira, F. C. N. and Wright, R. N.: 1997, Finite-state approximation of phrase-structure grammars, *in* E. Roche and Y. Schabes (eds), *Finite-State Language Processing*, MIT Press, pp. 149–173.

Schoorl, J. J. and Belder, S.: 1990, Computational linguistics at Delft: A status report, *Report WTM/TT 90–09*, Delft University of Technology, Applied Linguistics Unit.

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