Chapter 2

WEIGHTED GRAMMAR TOOLS: THE GRM LIBRARY

Mehryar Mohri
AT&T Labs — Research
180 Park Avenue, Rm E147
Florham Park, NJ 07932, USA
mohri@research.att.com

Abstract We describe the algorithmic and software design principles of a general grammar library designed for use in spoken-dialogue systems, speech synthesis, and other speech processing applications. The library is a set of general-purpose software tools for constructing and modifying weighted finite-state acceptors and transducers representing grammars.

The tools can be used in particular to compile weighted context-dependent rewrite rules into weighted finite-state transducers, read and compile, when possible, weighted context-free grammars into weighted automata, and dynamically modify the compiled grammar automata. The dynamic modifications allowed include: grammar switching, dynamic modification of rules, dynamic activation or non-activation of rules, and the use of dynamic lists. Access to these features is essential in spoken-dialogue applications.

2.1 Motivation

We describe the algorithmic and software design principles of a general grammar library (GRM library) designed for use in spoken-dialogue systems, speech synthesis, and other speech processing applications.

Most grammars used in speech processing applications are *weighted*. The use of grammar weights is crucial for building robust spoken-dialogue systems. In speech recognition, grammar weights are combined with acoustic weights to rank different hypotheses for a given speech utterance [Rabiner and Juang 1993, Jelinek 1998]. Due to the high variability of the input speech signal, a robust grammar admits any hypothesis as a possible transcription with a certain degree of probability.

Jean-Claude Junqua and Gertjan van Noord (eds), ROBUSTNESS IN LANGUAGE AND SPEECH TECHNOLOGY. Kluwer Academic Publishers, 2000, pp. 19–40.

Similarly, in speech synthesis (TTS) applications, weights are used to produce a single output even in presence of ambiguities. Grammar weights, weights of context-dependent morphological and phonemic rules, and other acoustic weights such as target and join costs in concatenative TTS systems [Beutnagel, Mohri and Riley 1999], are used to select the most plausible output speech signal. Thus, one crucial requirement for creating a grammar library for robust speech processing applications is that it allows one to create and use *weighted* rules and grammars.

The original goal of the GRM library was to provide algorithms and representations for constructing and using dynamic grammars in very large-vocabulary speech recognition. This led to the following requirements: generality, to support the representation and use of the various grammars in dynamic speech recognition, and efficiency, to support competitive large-vocabulary dynamic recognition using automata of more than 1M states and transitions.

Generality led us to define the most extended algebraic framework for each algorithm [1] as in a general-purpose finite-state machine library (FSM library) [Mohri, Pereira and Riley 2000]. The weight set for the rules and grammars used in the GRM library can be an arbitrary semiring. [2] The implementation given for each algorithm is very general: it does not depend on the underlying semiring used. A general implementation also helps avoiding redundancy and reducing the amount of code.

The principle of efficiency led us to design and implement the best known algorithm for each problem, and to carefully reduce the memory requirement of each implementation. This was achieved in particular by devising dynamic or on-the-fly algorithms such as in our compilation of context-free grammars. Only the necessary part of the resulting weighted automaton is constructed or expanded for the specific input sequence. We present and illustrate in more detail the application of these principles in the design of this library.

2.2 Overview

The GRM library is a set of general-purpose software tools for constructing and modifying weighted finite-state acceptors and transducers representing grammars.

The tools can be used in particular to: compile weighted context-dependent rewrite rules into weighted finite-state transducers, read and compile, when possible, weighted context-free grammars into weighted automata, and dynamically modify the compiled grammar automata. The dynamic modifications allowed include: grammar switching, dynamic modification of rules,

---

1. See Mohri [1998] for the definition of these frameworks and corresponding generic algorithms for shortest-distance problems.
2. A *semiring* is a ring that may lack negation.

dynamic activation or non-activation of rules, and the use of dynamic lists. Access to these features is essential in spoken-dialogue applications.

The GRM library is a high-level library based on the FSM library [Mohri et al. 2000] of which it is using some of the functions. The format of the automata and transducers used in the GRM library is compatible with that of the FSM library. This allows one to use numerous utilities available with the FSM library to manipulate the machines created by the GRM library. In particular, thanks to this compatibility, one can use general algorithms, such as ϵ-removal, ϵ-normalization, determinization, and minimization of weighted transducers to optimize the result of the compilation of weighted rewrite rules, or that of weighted context-free grammars.

The GRM tools are accessible through three levels.

User-program level

The GRM user-program level consists of a set of programs that manipulate grammars by reading from and writing to files or pipelines, and the GRM textual file format specification. For example, `grmcdcompile` returns a weighted finite-state transducer that represents the context-dependent rewrite rule given by the input FSMs [3].

C library level

The GRM C library level consists of a library archive of C functions, which implement the user program level. Typically, a program from the user-level (e.g., `grmcdcompile`) is implemented by a call to a corresponding library function (e.g., `GRMCdCompile`), preceded by a GRM or FSM library function call to read input rules (`GRMRead`) or an input FSM (`FSMLoad`) into an internal object, and followed by the FSM library function call to write out the resultant FSM (`FSMDump`).

Definition level

The GRM definition level consists of the definition level of the FSM library as well as other low-level definitions of some FSMClass objects defined in the GRM library. Accessing this level is important if one wishes to know the finer points or extend the internals of the GRM library.

2.3 Compilation of weighted rewrite rules

The library includes a complete set of routines and utilities to compile *weighted context-dependent rules* into weighted finite-state transducers.

3. We will sometimes use the term FSM to refer to finite-state machines given in the format of the FSM library.

2.3.1 Weighted context-dependent rules

Context-dependent rewrite rules are used in many areas of text and speech processing, including syntax, morphology, and phonology. As shown by Johnson [1972], rewrite rules can be represented by finite-state transducers, under the condition that no rule be allowed to apply any more than a finite number of times to its own output. Kaplan and Kay [1994], or equivalently Karttunen [1995], presented an algorithm for compiling rewrite rules into finite-state transducers, under the equivalent condition that they do not rewrite their non-contextual part. We will also assume that condition in what follows.

In some applications such as those related to speech processing, in particular in speech synthesis, the input may allow many possible analyses. One can then use weights to assign some likelihood or uncertainty measure to each of these analyses. Such generalized rules are called *weighted context-dependent rules*. More formally, weighted context-dependent rewrite rules have the following general form:

$$\phi \rightarrow \psi/\lambda \underline{\quad} \rho$$

They can be interpreted in the following way: ϕ is to be replaced by ψ whenever it is preceded by λ and followed by ρ. λ and ρ represent the left and right contexts of application of the rules. The rewriting condition mentioned previously means that after a replacement of ϕ by ψ, ψ can still serve as a context for further applications of the rule, but it cannot be rewritten.

In general, ϕ, λ and ρ are all regular expressions over the alphabet of the rules, and ψ a weighted regular expression, or a rational power series [Berstel and Reutenauer 1988, Salomaa and Soittola 1978]. The weight set for these power series can be any semiring $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$. In most cases, such as those considered in speech synthesis applications, the weights are interpreted as $-\log$ probabilities and a Viterbi approximation is assumed. The appropriate semiring $\mathbb{K}$ is then the tropical semiring $\mathcal{T} = (\mathbb{R}_+ \cup \{\infty\}, \min, +, \infty, 0)$.

Context-dependent rules can be used both in an optional and an obligatory mode depending on whether any occurrence of ϕ in the appropriate context is replaced by ψ (obligatory case) or not (optional case).

A rule can be applied from left to right, right to left, or in a simultaneous way. When applied from left to right, each replacement applies to the right of the output of the previous replacement. Similarly, in the case of a right-to-left application, it applies to the left of the output of the previous replacement. In the simultaneous case, the contexts of application of the rule are directly determined in the original string before any replacement.

As an example, the following weighted rule:

$$aa \rightarrow .25\ ba/a \underline{\quad} \epsilon$$

replaces each sequence of aa into ba with weight $.25$, when aa has left context a and right context ϵ. The effect of the rule on various strings is illustrated by

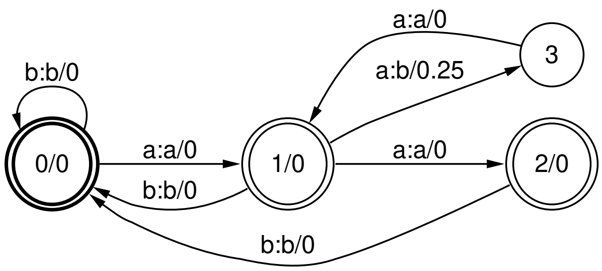


FIGURE 2.1: Weighted finite-state transducer realizing the weighted left-to-right obligatory rule $aa \rightarrow .25ba/a\_\_\epsilon$. The transducer is computed by the shell command: `grmcdcompile -i LAB AA BB A EPS`, where `AA`, `BB`, `A`, and `EPS` are automata representing aa, $.25\ ba$, a, and ϵ, and `LAB` the alphabet $\Sigma = \{a, b\}$. It has been further ϵ-removed and compacted using the utilities of the FSM library.

the following when it is interpreted as a left-to-right obligatory rule:

$$\begin{aligned} aaa &\rightarrow .25\,aba \\ aaaa &\rightarrow .25\,abaa \\ aaaaa &\rightarrow .50\,ababa \end{aligned}$$

Each application of the rule contributes weight $.25$ to the weight of the resulting string. By comparison, a right-to-left obligatory application of the rule gives:

$$\begin{aligned} aaa &\rightarrow .25\,aba \\ aaaa &\rightarrow .25\,aaba \\ aaaaa &\rightarrow .50\,ababa \end{aligned}$$

2.3.2 GRM utilities and illustration

Figure 2.1 shows the *weighted transducer* representing this rule when interpreted as left-to-right and obligatory. A weighted transducer is a labeled directed graph in which each transition is labeled with an input symbol, an output symbol and a weight in $\mathbb{K}$. A successful path is a path from an initial state (states in bold in the figure) to a final state (states represented by double circles in the figure). A successful path in a transducer pairs the concatenation of the input labels on its transitions with that of the corresponding output labels, assigning the pair the $\otimes$-product of the transition weights. The weight associated to a pair of input and output strings is obtained by taking the $\oplus$-sum of the weights of all successful paths labeled with that pair.

The transducer of figure 2.1 is obtained using the GRM utility `grmcd-compile` which compiles rules into weighted transducers. The syntax of the command is similar to the definition of the rules. `grmcdcompile` takes as

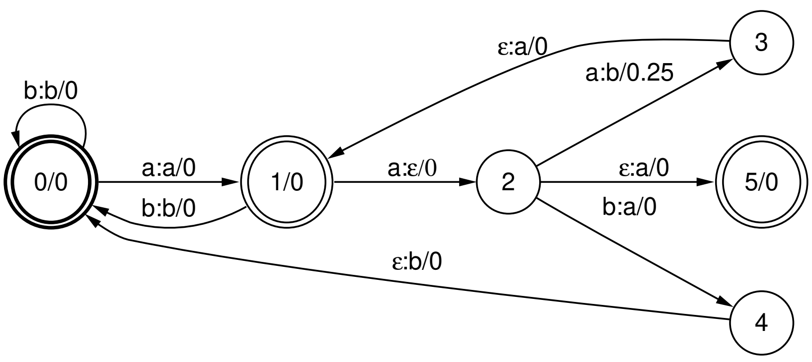


FIGURE 2.2: Subsequential transducer realizing the weighted left-to-right obligatory rule $aa \rightarrow .25\ ba/a\_\_\epsilon$ obtained from the transducer of figure 2.1 using the shell commands `fsmdeterminize`, `fsmpush` and `fsmcompact`. The transducer is subsequential and minimal with ϵ is considered as a regular symbol.

input four FSMs: f_1, f_2, f_3 and f_4, and returns a (weighted) transducer representing the context-dependent rewrite rule:

$$f_1 \rightarrow f_2/f_3\_\_f_4$$

The four input FSMs f_1, f_2, f_3 and f_4 must be acceptors. f_1, f_3 and f_4 must be costless. The rule expresses the fact that a string w recognized by f_1 is replaced by the (weighted) strings recognized by f_2 in the context where w is preceded by a string recognized by f_3 and followed by a string recognized by f_4.

The application of the rule proceeds from left to right by default or when the option `-l` is specified. It proceeds from right to left if `-r` is specified, and applies in parallel or simultaneously if the option `-s` is specified. If more than one option among these three is specified, all except the last one are ignored. The application of the rules is obligatory unless the option `-o` is specified, in which case the replacement is optional. The alphabet of the set of symbols of the rule is specified with the option `-i` symbols, which can be used to provide textual names for symbols. This feature can be used for instance to specify the initial non-terminal symbol with option `-s` using its textual name, S for instance, rather than the corresponding number associated to the non-terminal S.

Since the format of the finite-state machines produced by the GRM library is compatible with the FSM library, one can directly apply optimization algorithms such as weighted determinizaton and minimization to the output of `grmcdcompile`. The resulting minimal subsequential transducer is shown in figure 2.2. In some applications, it may be crucial to use the optimized transducer representing a weighted context-dependent rule both because of its efficiency of use and to save space.

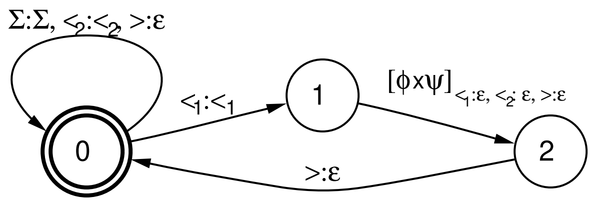


FIGURE 2.3: The construction of the transducer *replace*.

2.3.3 Algorithm

This section briefly presents and illustrates the compilation algorithm for weighted context-dependent rules implemented in the GRM library. We refer the reader to Mohri and Sproat [1996] for a more detailed description of the algorithm.

The algorithm is based on the use of *marking transducers*, that is transducers that introduce or delete auxiliary symbols marking either the beginning or the end of sequences described by a regular expression. These transducers are used to introduce auxiliary symbols only when necessary as opposed to the algorithm of Kaplan and Kay [1994] (KK for short) which introduces brackets everywhere only to restrict their occurrence subsequently. This makes intermediate results in the compilation of the rules easier to read and check. The number of intermediate transducers necessary in the construction of the rules is also smaller than in the KK algorithm, and each of the transducers can be constructed more directly and efficiently from the primitive expressions of the rule, ϕ, ψ, λ, ρ.

The construction of the marking transducers consists of the determinizations of three automata and other work linear in the size of those automata. The algorithm requires fewer compositions than the KK algorithm. On-the-fly composition of weighted transducers [Mohri, Pereira and Riley 1996] can be used to construct only the part of the compilation result needed for the specific input sequence considered. The algorithm was shown to be more efficient than the KK algorithm [Mohri et al. 1996].

For the sake of simplicity, we illustrate the algorithm in the case of an obligatory left-to-right rule:

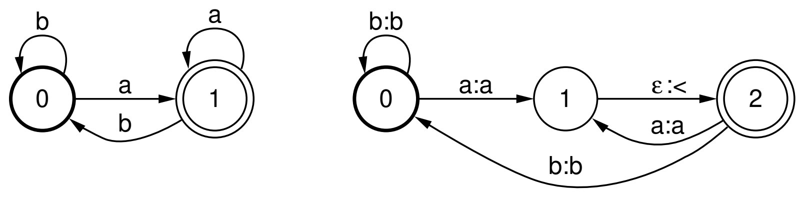
$$\phi \to \psi / \lambda \_\_\_ \rho$$

Other combinations (optional, right-to-left, or simultaneous) can be easily derived from this case.

The following composition cascade gives a brief description of the compilation algorithm:

$$T = r \circ f \circ \mathit{replace} \circ l_1 \circ l_2$$

where r, f, *replace*, l_1, and l_2 are weighted transducers defined as follows. Transducer r is a marking transducer introducing an auxiliary left bracket symbol $>$ ($>$ does not belong to the alphabet Σ) before each occurrence of a se-



(A) (B)

FIGURE 2.4: (A) Minimal deterministic automaton representing $\alpha = (a+b)^* a$. (B) Transducer B post-marking by $<$ each occurrence of a string described by α.

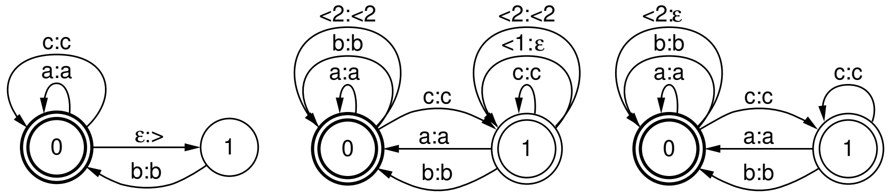


FIGURE 2.5: Marking transducers r, l_1, and l_2.

quence accepted by the regular expression ρ in the input. r can be described by the following left-to-right obligatory context-dependent rule:

$$r : \epsilon \to > /\Sigma^* \underline{\quad} \rho$$

The transducer f is a marking transducer introducing auxiliary right bracket symbols $<_1$ and $<_2$ before each occurrence of a sequence accepted by ϕ and followed by $>$ in a string over the alphabet $\Sigma \cup \{>\}$ with the auxiliary symbol $>$ freely appearing at any position ($>$ is ignored for matching purposes). The corresponding transduction is given by the left-to-right obligatory context-dependent rule over the alphabet $\Sigma \cup \{>\}$:

$$f : \epsilon \to \ (\{<_1\} \cup \{<_2\})/(\Sigma \cup \{>\})^* \underline{\quad} \phi_> \ >$$

where $\phi_>$ is the regular expression obtained from ϕ by freely allowing the symbol $>$ anywhere in ϕ. The use of $\phi_>$ instead of ϕ is necessary since $>$ should be ignored in the determination of the occurrences of ϕ.

Thus $r \circ f$ transforms an input string by introducing brackets (left bracket $<_1$ and $<_2$ and right bracket $>$) around sequences accepted by ϕ that are followed by a sequence accepted by ρ. The transducer *replace* replaces each of such ϕ sequences by all weighted sequences given by the rational power series ψ, additionally erasing the right bracket $>$. Figure 2.3 illustrates the construction of *replace*. The symbols $>$, $<_1$, and $<_2$ need to be ignored when determining an occurrence of ϕ, thus there are loops over the transitions

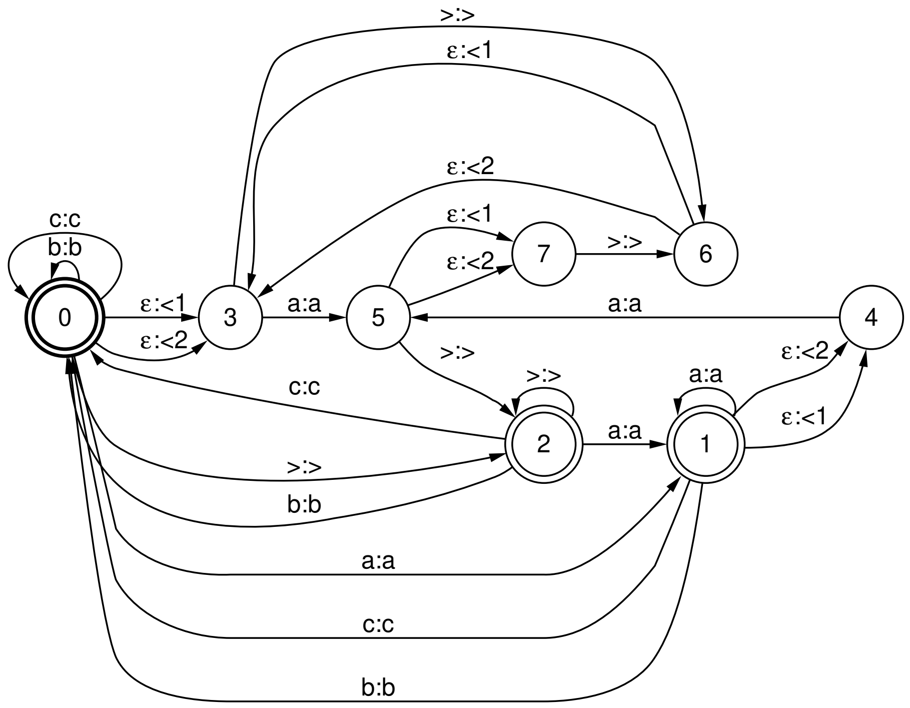


FIGURE 2.6: Transducer f.

$>: \epsilon, <_1: \epsilon, <_2: \epsilon$ at all states of the automaton accepting ϕ, or equivalently of the states of transducer realizing the cross product $\phi \times \psi$.

l_1 is a marking transducer that accepts only strings where a sequence accepted by λ is followed by $<_1$ and it erases that symbol in those strings. l_1 can be described by the context-dependent rule:

$$l_1 : <_1 \rightarrow \epsilon / \# \Sigma^* \lambda \_\_\_ \epsilon$$

where $\#$ denotes the beginning of the input sequence, with the symbol $<_2$ ignored everywhere in the input.

Similarly, the transducer l_2 is a marking transducer that only accepts strings where $<_2$ is preceded by a sequence not accepted by λ, namely a sequence accepted by the complement of $\Sigma^* \lambda$ denoted by $\overline{\Sigma^* \lambda}$, and erases that auxiliary symbol in those strings:

$$l_2 : <_2 \rightarrow \epsilon / \# \overline{\Sigma^* \lambda} \_\_\_ \epsilon$$

Thus $l_1 \circ l_2$ accepts only those strings where $<_1$ follows a sequence recognized by λ and $<_2$ a sequence not recognized by λ and erases these auxiliary symbols.

It is clear from the definition of these transducers that T realizes the weighted transduction defined by the weighted context-dependent rule. Note

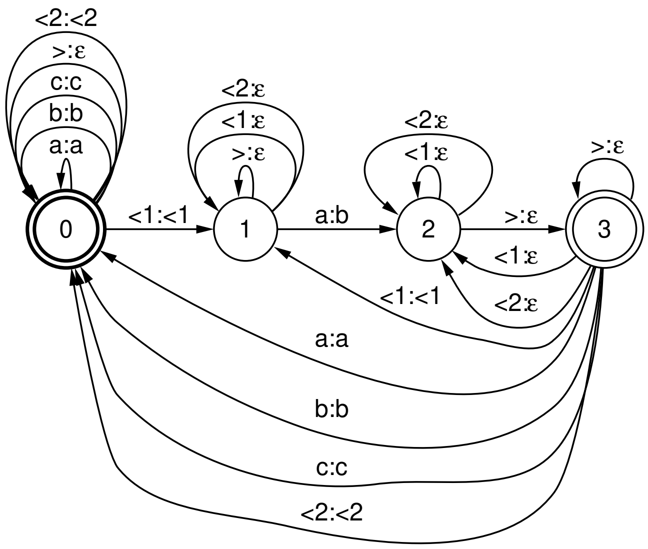


FIGURE 2.7: Transducer *replace*.

that the weighted transducer *replace* is independent of the left and right contexts λ and ρ, and thus can be computed independently. The construction of a marking transducer for a regular expression is based on a deterministic automaton representing that expression. We illustrate this by describing the construction of a marking transducer corresponding to the transduction:

$$\Sigma^* \alpha \rightarrow \Sigma^* \alpha <$$

where α is an arbitrary regular expression over the alphabet Σ. Let $A = (\Sigma, Q, I, F, E)$ be a deterministic automaton recognizing α, where Σ is the alphabet, Q the state set, $I \subseteq Q$ the set of initial states, $F \subseteq Q$ the set of final states and E the finite set of transitions of A.

A can be viewed as a transducer if we assign to each transition an output label equal to the input label. We can modify the corresponding transducer so that it inserts the symbol $<$ after each occurrence of a string accepted by α as follows:

1. all states in $Q - F$ are made final,
2. for each state state $q \in F$
   (a) a new state q' copy of q is created (q' has the same transitions as q, and q' is a final state)
   (b) q is made non-final,
   (c) outgoing transitions of q are removed, and replaced by a single transition from q to q' with input label ϵ and output $<$.

Figure 2.4 illustrates the construction of the resulting transducer B for the specific case of $\alpha = (a+b)^* a$. Figures 2.5, 2.6, and 2.7 give explicitly the transduc-

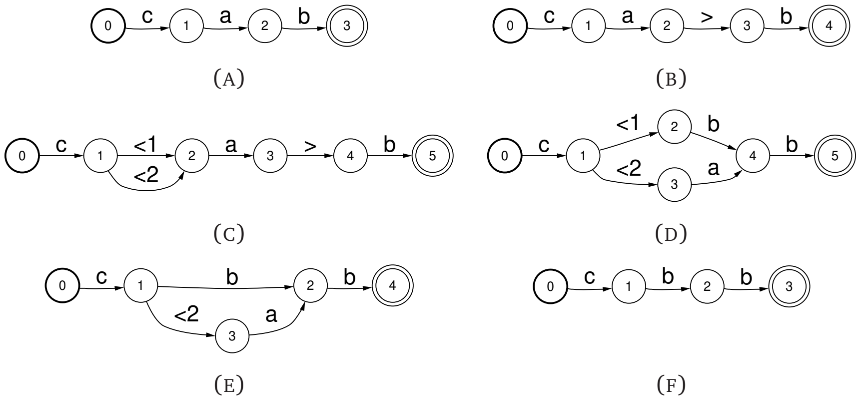


FIGURE 2.8: Intermediate results after application of each transducer of the rule $a \to b/c\_\_b$ to the input string cab. (A) Input automaton recognizing cab. (B) Result of the application of the transducer r. (C) Result after application of f. (D) Result after application of *replace*. (E) Result after application of l_1. (F) Final output resulting from the application of l_2.

ers used in the compilation of the rule $a \to b/c\_\_b$. The successive applications of these transducers to the input string cab are illustrated by figures 2.8 (a)–(f).

2.4 Compilation of weighted context-free grammars

The GRM library includes a general set of algorithms and utilities to compile weighted context-free grammars. The compilation algorithm can be executed on demand, as needed for the recognition of the actual input utterances, and the grammars can be dynamically modified.

2.4.1 Weighted context-free grammars

Context-free grammars (CFGs) are widely used in text and speech processing applications. In some applications such as speech recognition, it is necessary to use *weighted* CFGs by assigning some weights to grammar rules. These weights, often interpreted as probabilities, are then combined with acoustic weights to rank recognition transcriptions.[4]

The weight set of the weighted context-free grammar used can be any semiring $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$. In most cases, such as those considered in speech recognition applications, the weights are interpreted as $-\log$ probabilities and a Viterbi approximation is assumed. The appropriate semiring $\mathbb{K}$ is then the tropical semiring $\mathcal{T} = (\mathbb{R}_+ \cup \{\infty\}, \min, +, \infty, 0)$. For example, the following weighted CFG:

4. See Bahl, Jelinek and Mercer [1983] and Mohri and Riley [1999] for a full description of the way the grammar or language model can be combined with other components of a recognition system – pronunciation dictionary, phonemic context-dependency model, acoustic model.

$$\begin{array}{rcl} Z\ .1 & \rightarrow & XY \\ X\ .2 & \rightarrow & aY \\ Y\ .3 & \rightarrow & bX \\ Y\ .4 & \rightarrow & c \end{array}$$

FIGURE 2.9: Simple CFG grammar G.

assigns weight .1 to the rule $Z \rightarrow XY$, and weight .2 to $X \rightarrow aY$.

A weighted CFG $G = (V, P)$ over the alphabet Σ, with weights in a semiring $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ consists of a finite alphabet V of variables or nonterminals disjoint from Σ, and a finite set $P \subseteq V \times \mathbb{K} \times (V \cup \Sigma)^*$ of productions or derivation rules [Autebert, Berstel and Boasson 1997]. Given strings $u, v \in (V \cup \Sigma)^*$, and weights c and c', we write $(u, c) \stackrel{*}{\rightarrow} (v, c')$ when there is a derivation from u with weight c to v with weight c'. We denote by $L_G(X)$ the weighted language generated by a nonterminal X:

$$L_G(X) = \{(w, c) \in \Sigma^* \times \mathbb{K} : (X, 0) \stackrel{*}{\rightarrow} (w, c)\}$$

Context-free grammars used in spoken-dialogue applications often represent regular languages [Church 1983, Brown and Buntschuh 1994], either by construction or as a result of a finite-state approximation of a more general CFG [Pereira and Wright 1991, Nederhof 1998, Mohri and Nederhof 2000]. This is because in most real-time applications, general CFGs are computationally too demanding.

The GRM library provides general tools for compiling a weighted CFG into a weighted finite automaton under certain general conditions described below. In addition, our compilation method allows the recognizer to dynamically *activate* or *deactivate* grammar rules and *substitute* a new regular language for some terminal symbols, depending on previously recognized inputs, all without recompilation. This is the terminology used in this field, a more appropriate expression would be dynamic activation or deactivation of *nonterminal symbols*. The following paragraphs define more precisely these dynamic modifications.

Dynamic activation or deactivation of rules

We define a set of *active nonterminals* which are the start symbols for derivations. Let $A \subseteq V$ be the set of active nonterminals, then the language generated by G is:

$$L_G = \bigcup_{X \in A} L_G(X)$$

Note that inactive nonterminals, and the rules involving them, can be used in derivations, they are just not available as start symbols. Dynamic rule activa-

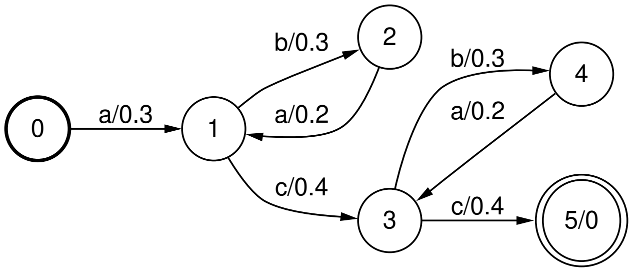


FIGURE 2.10: Weighted finite automaton representing the weighted CFG G (figure 2.9). The automaton is computed by the shell command:

```
grmread -i LAB -w R | grmcfcompile -i LAB -s Z
```

where LAB is the alphabet $\Sigma = \{a, b, c, X, Y, Z\}$, R the textual representation of the CFG. It has been further ϵ-removed and determinized and minimized using the utilities of the FSM library.

tion or deactivation consists exactly of the dynamic redefinition of the set A in successive uses of the grammar.

Dynamic substitution

Let σ be a weighted rational transduction of Σ^* to $\Delta^* \times \mathbb{K}$, $\Sigma \subseteq \Delta$, that is a regular weighted substitution [Berstel 1979]. σ is a monoid morphism verifying:

$$\forall x \in \Sigma, \sigma(x) \subseteq Reg(\Delta^* \times \mathbb{K})$$

where $Reg(\Delta^* \times \mathbb{K})$ denotes the set of weighted regular languages over the alphabet Δ. Thus σ simply substitutes for each symbol $a \in \Sigma$ a weighted regular expression $\sigma(a)$. A dynamic substitution consists of the application of the substitution σ to Σ, during the process of recognition of a word sequence. Thus, after substitution, the language generated by the new grammar G' is:[5]

$$L_{G'} = \sigma(L_G)$$

The GRM compilation algorithm allows for both of those dynamic grammar changes without recompiling the grammar.

2.4.2 GRM utilities and illustration

The textual representation R of the grammar G of figure 2.9 in the GRM library is simply:

```
Z 1 X Y
X 2 a Y
Y 3 b X
```

5. σ can be extended as usual to map $\Sigma^* \times \mathbb{K}$ to $Reg(\Delta^* \times \mathbb{K})$.

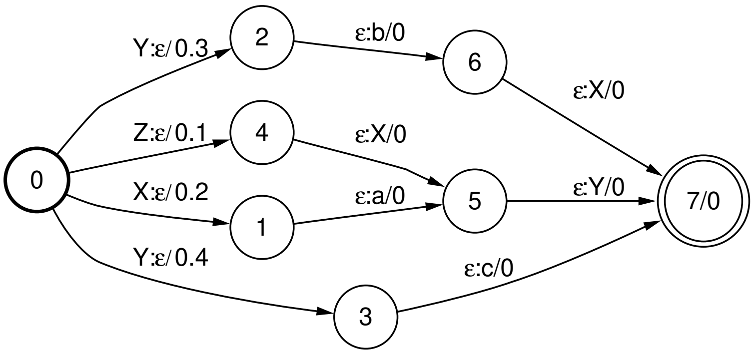


FIGURE 2.11: Weighted transducer output of `grmread`.

```
Y 4 c
```

Figure 2.10 shows the weighted automaton result of the compilation of the weighted CFG G with start symbol Z using the GRM utilities `grmread` and `grmcfcompile`.

`grmread` takes as input a textual representation of a weighted context-free grammar from file or standard input, and sends to standard output the binary encoding of a weighted transducer representing the grammar. The grammar can be represented using compound rules (by factorization of the left-hand side of the rules), if the option `-c` is used. For example, the following is a more compact representation of the grammar G:

```
Z 1 X Y
X 2 a Y
Y 3 b X | 4 c
```

Input symbols are represented in the input file by non-negative numbers, unless the option `-i` is used. This option allows non-terminal and terminal symbols to be given textual names, where the file symbols gives the translation from names to numbers. The input must be an unweighted context-free grammar, unless the option `-w` is given, in which case it must be a weighted context-free grammar.

The representation of the grammar as a weighted transducer is crucial because it can then be optimized (compacted, or determinized and minimized [Mohri 1997, Mohri 2000]). Such a preprocessing dramatically improves the time and space required by the main compilation step thanks to increased sharing among grammar rules that start or end the same way [Mohri and Pereira 1998]. Figure 2.11 shows the output of the `grmread` for the grammar G after compaction.

`grmcfcompile` accepts as input a weighted transducer representing a context-free grammar and returns a delayed weighted acceptor recognizing that grammar. The input transducer must represent a context-free grammar for a weighted regular language. More precisely, the restriction on the rules of the context-free grammar is the following: the rules of a set of non-terminals that are mutually dependent (a set $\{X, Y\}$ for instance, with X depending on Y and Y depending on X) must be either right-linear or left-linear with respect to the elements of this set. For example, the following grammar:

$$\begin{array}{lcl} X & \rightarrow & aRY \\ Y & \rightarrow & bTVX \\ R & \rightarrow & cVTd \\ V & \rightarrow & TV \\ Y & \rightarrow & e \\ T & \rightarrow & f \\ V & \rightarrow & g \end{array}$$

is well-formed. The non-terminals X and Y depend on each other, and the rules defining X and Y are right-linear with respect to X and Y. The grammar below:

$$\begin{array}{lcl} X & \rightarrow & aRY \\ Y & \rightarrow & bTVX \\ R & \rightarrow & cVTd \\ V & \rightarrow & dX \\ Y & \rightarrow & e \\ T & \rightarrow & f \\ V & \rightarrow & g \end{array}$$

is not well-formed because X, Y, R and V depend on each other, but the rule $X \rightarrow aRY$ for instance, is neither left-linear nor right-linear with respect to these symbols.

The start symbol of the grammar can be specified using the option `-s`. The option `-i` is used to provide a textual name for the non-terminal symbol. This option is also used for error messages. The list of the non-terminals that are problematic is printed out when the input grammar is not well-formed.

When the grammar is made of only right-linear rules, or only left-linear rules, a more efficient compilation algorithm can be used. The option `-l` of `grmcfcompile` can be used for that purpose.

The object returned by `grmcfcompile` is a *delayed* acceptor which admits any FSM operation valid for acceptors. Only the necessary part of that acceptor (states and transitions) is expanded when using an FSM operation.

The delayed objects created by the GRM library are based on two FSM classes: the Edit class and the Replace class. [6] These are essential features that allow one to dynamically modify the input grammar.

6. In particular, the acceptor returned by `grmcfcompile` belongs to the Replace class.

Edit class

The Edit class is an FSM class defined by the GRM library. It allows users to dynamically modify the initial state of an FSM or the final cost of a state of an FSM. An Edit class object represents an FSM F defined by an input FSM S and a new initial state s, a state f, and a final cost c.

The FSM F is exactly the machine obtained from S by the following delayed operations: the state s becomes the new initial state, and the final cost c is assigned to f.

Replace class

The Replace class is an FSM class defined by the GRM library. It allows users to construct FSMs with labels referring to other FSMs. A Replace class object represents an FSM F defined by an FSM S that gives the general structure of F and an array of FSMs, $T[i]$, indexed with the set of all symbols i including the input labels of S.

The FSM F is exactly the result of the replacement by $T[i]$ of each label i of S for which $T[i]$ is defined. These replacements are done on-demand for a given input. In the case of Replace class acceptors representing context-free grammars, typically $T[i]$ is defined for each i corresponding to a non-terminal symbol, and S is an acceptor accepting exactly the set of initial non-terminals of the grammar.

Dynamic modifications

Dynamic modifications of the input grammar are essential in many spoken-dialogue systems. Indeed, those systems require grammars or language models to evolve as the dialogue proceeds, to exploit the information and context provided by previous interactions.

For instance, a previous request for an address would activate the address grammar and lexicon and inactivate the date grammar and lexicon in an automated reservations task. The use of such dynamic grammars can significantly improve the efficiency and accuracy of the system by considerably restricting the set of possible transcriptions.

The result of the compilation of CFG can be directly modified at the command level by the following dynamic modification utilities: `grmreplace`, `grmmutate`, and `grmsubstitute`. Note that these modifications do not require a recompilation of the grammar. Their results, just as their inputs, are delayed objects of the Replace class. They are possible thanks to the definition of the Replace class.

`grmreplace` takes an input finite-state machine S and returns an FSM F that belongs to the GRM Replace class.

`grmmutate` takes as input two FSMs f_1 and f_2 and returns the FSM result of the replacement in f_2 of the structure FSM by f_1. A typical application of `grmmutate` is the dynamic modification of the initial non-terminal symbols

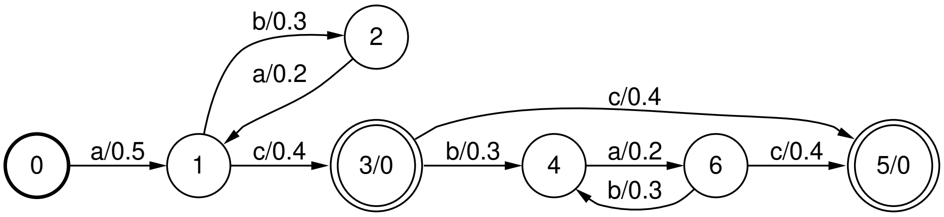


FIGURE 2.12: Dynamically modified grammar G with active non-terminals Z and X and weights .2 and .3. The automaton was generated by the following shell command:

```
grmmutate A F
```

where F is the delayed object output of `grmcfcompile` applied to the grammar G and A the activation acceptor.

of a context-free grammar (also called dynamic activation or inactivation of the rules of a context-free grammar) as defined above. Figure 2.12 illustrates the use of the `grmmutate` and the corresponding output automaton.

`grmsubstitute` takes as input a substitution file and a finite-state machine f and returns the FSM obtained by substituting FSMs for the symbols of f using the mapping defined in the substitution file. A typical application of `grmsubstitute` is the dynamic replacement of the terminal symbols of a context-free grammar (also called use of dynamic lists).

The GRM library has been used successfully to compile and dynamically modify context-free grammars of several hundred thousand rules.

2.4.3 Algorithm

This section briefly presents and illustrates the compilation algorithm for weighted CFGs as implemented in the GRM library. We refer the reader to Mohri and Pereira [1998] for a more detailed description of the algorithm.

The algorithm first gives a transducer representation of the input grammar which can be determinized and minimized to take advantage of more sharing among rules. The transducer representation is then used to:

1. build the dependency graph D_G of the input grammar G, that is the graph that has a node corresponding to each non-terminal and that has an edge from node X to node Y if non-terminal Y appears on the right hand-side of a rule with left hand-side X;
2. compute the strongly connected components of D_G, namely the sets of mutually dependent non-terminals; [7]
3. assuming that the grammar reduced to each SCC is either left-linear or right-linear, a weighted automaton $K(S)$ can be constructed for each

7. Recall that the strongly connected components, SCCs, of a directed graph are the equivalence classes of its nodes under the relation R defined by: $q\ R\ q'$ if q' can be reached from q and q from q' [Tarjan 1972].

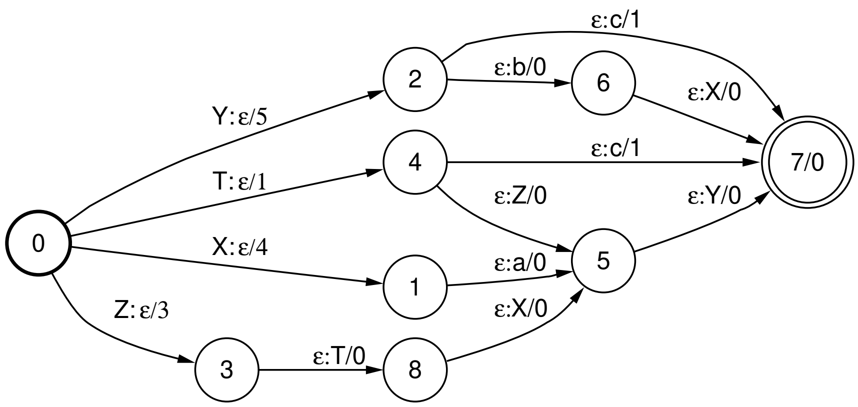


FIGURE 2.13: Compact transducer representation of the grammar G'.

SCC S, and for each non-terminal $X \in S$ a weighted automaton $M(X)$ can be built from $K(S)$; [8]

4. a simple automaton M_G accepting exactly the set of active nonterminals A is constructed;
5. the automaton is then expanded on-the-fly for each input string using lazy replacement and editing.

The consecutive steps of the compilation algorithm are illustrated for the following CFG:

$$
\begin{array}{lcl}
T\,1 & \rightarrow & ZY \\
T\,2 & \rightarrow & c \\
Z\,3 & \rightarrow & TXY \\
X\,4 & \rightarrow & aY \\
Y\,5 & \rightarrow & bX \\
Y\,6 & \rightarrow & c
\end{array}
$$

FIGURE 2.14: Grammar G'.

Figure 2.13 shows the representation of G' as a weighted transducer and the benefit of transducer determinization and minimization for sharing among rules starting or ending in the same way.

Figure 2.15 gives the dependency graph of the grammar G' which has two SCCs, S_1 and S_2.

8. The construction of each $M(X)$ from $K(S)$ is a delayed operation based on the Edit class, since $M(X)$ is derived from $K(S)$ just by picking out the state associated to X in $K(S)$.

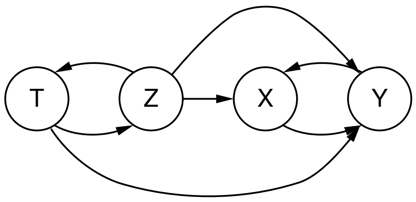
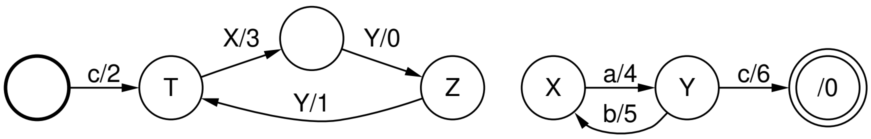


FIGURE 2.15: Dependency graph $D_{G'}$ of the grammar G'. The strongly connected components of $D_{G'}$ are $S_1 = \{T, Z\}$ and $S_2 = \{X, Y\}$.



(A) (B)

FIGURE 2.16: (A) Weighted automaton $K(S_1)$ for the SCC $S_1 = \{T, Z\}$. (B) Weighted automaton $K(S_2)$ for the SCC $S_2 = \{X, Y\}$.

The construction of automata representing left-linear or right-linear grammars is classical [Aho, Hopcroft and Ullman 1974]. That construction can be easily extended to weighted grammars to build the weighted automata corresponding to the SCCs S_1 and S_2, see figure 2.16 (a) and (b).

Our compilation algorithm works only with the class of grammars such that the grammar restricted to each SCC is either left-linear or right-linear. This seems to cover a large set of useful grammars. Since there is no general algorithm for mapping an arbitrary CFG generating a regular language into a corresponding finite automaton [Ullian 1967], the use of a restriction is necessary.

The automaton corresponding to each non-terminal can be easily constructed from $K(S_1)$ and $K(S_2)$ by making final (left-linear case) or initial (right-linear case) the corresponding state of $K(S_1)$ or $K(S_2)$. As mentioned above, in our implementation, this operation is delayed. The automaton $M(T)$ for instance is only constructed and expanded if needed for the specific input string to parse.

The automaton $M_{G'}$ accepts exactly the set of active non-terminals. Figure 2.17 shows that automaton for the specific case of $A = \{T, X\}$. Note that this automaton can be dynamically modified without requiring a recompilation of the grammar (the steps 1-3 of the algorithm do not need to be repeated).

The automaton $M_{G'}$ can then be expanded for each input string using lazy replacement and editing: each non-terminal symbol B is replaced on-the-fly by its corresponding automaton $M(B)$ which is itself computed by a delayed editing operation of the automaton $K(S)$ with $B \in S$. Figure 2.18 shows the

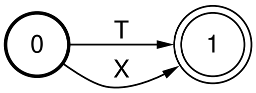


FIGURE 2.17: Automaton $M_{G'}$ accepting the set of active non-terminals $A = \{T, X\}$.

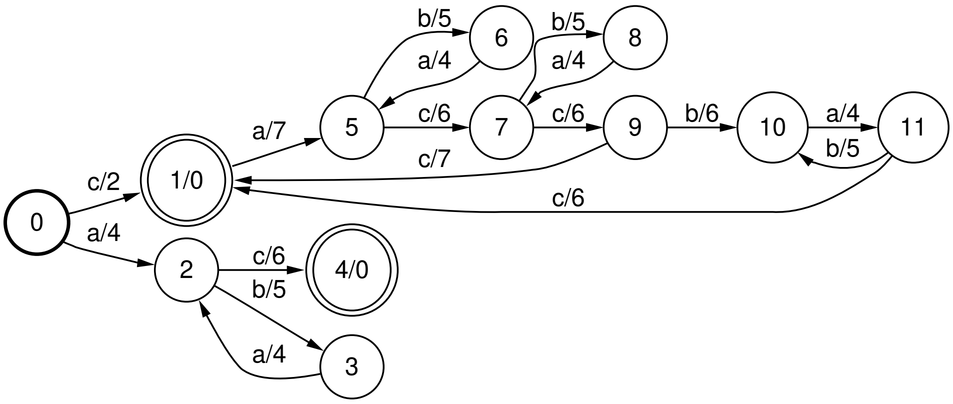


FIGURE 2.18: Automaton $M_{G'}$ after full expansion.

fully expanded automaton $M_{G'}$ after removal of ϵ's, determinization, and minimization.

Our experiments both with full bigram models of various vocabulary sizes, and with unweighted grammars derived by feature instantiation from hand-built feature-based grammars showed the efficiency of the algorithm [Mohri and Pereira 1998].

2.5 Conclusion

We presented a general weighted grammar library designed for use in spoken-dialogue systems, speech synthesis, and other speech processing applications. Since it is based on the notions of semiring and of rational power series, the same code can be used for a variety of different applications. The current version of the library is written in C, with the semiring operations defined in the FSM library.

The library is tested with randomly generated automata. Its efficiency was tested by compiling and dynamically modifying successfully very large grammars of more than several hundred thousand rules.

The GRM library is currently used in many projects and is included in the AT&T product WATSON. It is planned to be made available for academic use in the near future [Mohri 1996]. It includes other utilities such as *local determinization* that can be useful in the context of speech recognition projects.

The compilation algorithm and programs for context-dependent rules have been shown to be efficient in several tasks. It is used by many in Lucent Bell Laboratories to develop TTS text analyzers for several languages including Spanish, Italian, French, Romanian, German, Russian, Mandarin and Japanese. It was also used in AT&T Labs – Research to construct efficiently several components of a speech understanding system based on finite-state transducers. The GRM utilities for compiling weighted context-dependent rules can also be used to compile decision trees into weighted transducers [Sproat and Riley 1996] and build general context-dependent model transducers for speech recognition systems.

The compilation algorithms and tools for weighted context-free grammars have been used in various speech processing projects including the construction of spoken dialog systems and speech translation systems at AT&T Labs – Research.

Future versions of the GRM library will include a compiler for rational power series, algorithms for constructing and compiling weighted approximations of weighted context-free grammars [Mohri and Nederhof 2000], and transducer learning algorithms and tools. Because of their generality, the Replace and Edit classes presented above will be directly incorporated in the FSM library in future versions of both libraries.

Acknowledgments

I thank Richard Sproat, Michael Riley, and Fernando Pereira for discussions about some of the utilities of this library, and Ted Roycraft and Bruce Buntschuh for their comments and help with defining the dynamic grammar features. Many of the GRM programs were kindly tested by Richard Sproat, Ted Roycraft, and Bruce Buntschuh in the context of various text and speech processing applications.

References

Aho, A. V., Hopcroft, J. E. and Ullman, J. D.: 1974, *The Design and Analysis of Computer Algorithms*, Addison Wesley, Reading, MA.

Autebert, J.-M., Berstel, J. and Boasson, L.: 1997, Context-free languages and pushdown automata, *in* G. Rozenberg and A. Salomaa (eds), *Handbook of Formal Languages*, Vol. 1, Springer, pp. 111–172.

Bahl, L. R., Jelinek, F. and Mercer, R.: 1983, A maximum likelihood approach to continuous speech recognition, *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* **5**(2), 179–190.

Berstel, J.: 1979, *Transductions and Context-Free Languages*, Teubner Studienbucher, Stuttgart.

Berstel, J. and Reutenauer, C.: 1988, *Rational Series and Their Languages*, Springer-Verlag, Berlin, New York.

Beutnagel, M., Mohri, M. and Riley, M.: 1999, Rapid unit selection from a large speech corpus for concatenative speech synthesis, *Proceedings of the 6th European Conference on Speech Communication and Technology (Eurospeech '99)*, Budapest, Hungary.

Brown, M. K. and Buntschuh, B. M.: 1994, A context-free grammar compiler for speech understanding systems, *Proceedings of the International Conference on Spoken Language Processing (ICSLP '94)*, Yokohama, Japan, pp. 21–24.

Church, K. W.: 1983, A finite-state parser for use in speech recognition, *21st Meeting of the Association for Computational Linguistics (ACL '83), Proceedings of the Conference*, ACL.

Jelinek, F.: 1998, *Statistical Methods for Speech Recognition*, The MIT Press, Cambridge, MA.

Johnson, C. D.: 1972, *Formal Aspects of Phonological Description*, Mouton, Mouton, The Hague.

Kaplan, R. M. and Kay, M.: 1994, Regular models of phonological rule systems, *Computational Linguistics* **20**(3).

Karttunen, L.: 1995, The replace operator, *33rd Meeting of the Association for Computational Linguistics (ACL 95), Proceedings of the Conference, MIT*, ACL, Cambridge, MA.

Mohri, M.: 1996, *General-Purpose Weighted Grammar Tools: The GRM Library*, AT&T Labs — Research. http://www.research.att.com/sw/tools/grm.

Mohri, M.: 1997, Finite-state transducers in language and speech processing, *Computational Linguistics* **23**(2).

Mohri, M.: 1998, General algebraic frameworks and algorithms for shortest-distance problems, *Technical Memorandum 981210-10TM*, AT&T Labs - Research, 62 pages.

Mohri, M.: 2000, Minimization algorithms for sequential transducers, *Theoretical Computer Science* **234**.

Mohri, M. and Nederhof, M.-J.: 2000, Regular approximation of context-free grammars through transformation, *in* J.-C. Junqua and G. van Noord (eds), *Robustness in Language and Speech Technology*, Kluwer Academic Publishers, The Netherlands. This volume.

Mohri, M. and Pereira, F. C. N.: 1998, Dynamic compilation of weighted context-free grammars, *36th Meeting of the Association for Computational Linguistics (ACL '98), Proceedings of the Conference*, ACL, Montréal, Québec, Canada.

Mohri, M., Pereira, F. C. N. and Riley, M.: 1996, Weighted automata in text and speech processing, *Proceedings of the 12th biennial European Conference on Artificial Intelligence (ECAI-96), Workshop on Extended finite state models of language*, ECAI, Budapest, Hungary.

Mohri, M., Pereira, F. C. N. and Riley, M.: 2000, The design principles of a weighted finite-state transducer library, *Theoretical Computer Science* **231**, 17–32.

Mohri, M. and Riley, M.: 1999, Integrated context-dependent networks in very large vocabulary speech recognition, *Proceedings of the 6th European Conference on Speech Communication and Technology (Eurospeech '99)*, Budapest, Hungary.

Mohri, M. and Sproat, R.: 1996, An efficient compiler for weighted rewrite rules, *34th Meeting of the Association for Computational Linguistics (ACL '96), Proceedings of the Conference*, ACL, Santa Cruz, CA.

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.

Pereira, F. C. N. and Wright, R. N.: 1991, Finite-state approximation of phrase structure grammars, *29th Annual Meeting of the Association for Computational Linguistics (ACL '91), Proceedings of the Conference*, ACL, Berkeley, CA.

Rabiner, L. and Juang, B.-H.: 1993, *Fundamentals of Speech Recognition*, Prentice-Hall, Englewood Cliffs, NJ.

Salomaa, A. and Soittola, M.: 1978, *Automata-Theoretic Aspects of Formal Power Series*, Springer-Verlag, New York.

Sproat, R. and Riley, M.: 1996, Compilation of weighted finite-state transducers from decision trees, *34th Meeting of the Association for Computational Linguistics (ACL '96), Proceedings of the Conference, Santa Cruz, California*, ACL.

Tarjan, R. E.: 1972, Depth first search and linear graph algorithm, *SIAM Journal of Computing* **1**(2), 146–160.

Ullian, J. S.: 1967, Partial algorithm problems for context free languages, *Information and Control* **11**, 90–101.