9

Local Grammar Algorithms

MEHRYAR MOHRI

Local syntactic constraints can often be described with much precision. For example, the sequences of preverbal particles in French, e.g., *ne, le, lui*, obey strict rules that govern their ordering and the insertion of other terms among them, regardless of the remaining material forming the sentence (Gross, 1968). As such, they can be viewed as *local rules*, or *local grammars*. Similar detailed local grammars have been given for time expressions (Maurel, 1989) and later for many other linguistic expressions (Gross, 1997).

Such *local grammars* do not just capture some rare linguistic phenomena. Widespread technical jargon, idioms, or clichés, lead to common syntactic constraints that can be accurately described locally without resorting to more powerful syntactic formalisms. A careful examination of articles written in the financial section of a newspaper reveals for example that only a limited number of constructions accounts for the description of the variations of the stock market, or the changes in inflation or unemployment rate.

A local grammar may describe a set of forbidden or unavoidable sequences. In both cases, it can be compactly represented by a finite automaton. A collection of local grammars can be combined and represented by a more complex finite automaton by taking the union of the simpler local grammar automata. Novel linguistic studies keep increasing the number of local grammars (Gross, 1997). This tends to significantly increase the size of the union local grammar finite automata and creates the need for efficient algorithms to apply large local grammar automata.

This chapter presents an overview of algorithms for the application of local grammar automata. Section 1 introduces the algorithmic problem related to the application of large local grammar automata. Section 2 reviews two

Inquiries into Words, Constraints and Contexts.
Antti Arppe et al. (Eds.)
Copyright © 2005, by individual authors.

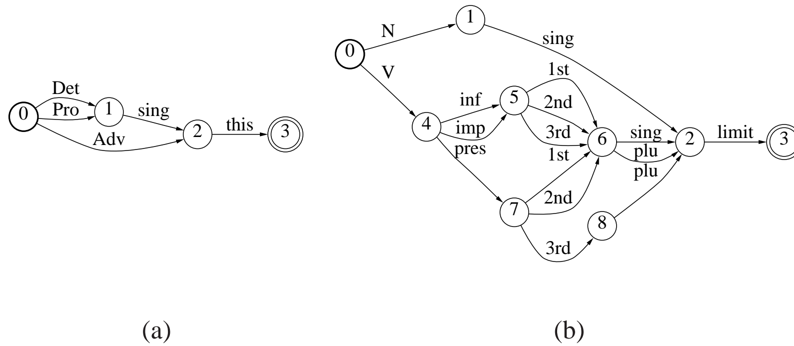

FIGURE 1 Automata representing the possible part-of-speech tags for (a) *this*; and (b) *limit*; in the absence of any context.

local grammar algorithms and presents in detail the most efficient one. It also illustrates these algorithms by showing examples of their applications.

9.1 Description of the problem

Let Σ denote the alphabet and let A be a local grammar finite automaton specifying a set of forbidden sequences. We denote by $L(A)$ the language accepted by A. By definition, any sequence containing a sequence accepted by A is unacceptable. Thus, acceptable sequences must be in $\overline{\Sigma^* L(A) \Sigma^*}$.

Let us illustrate this with an example related to part-of-speech tagging. Let T be the automaton representing the set of all possible tagging of a text (Koskenniemi, 1990). T can be obtained by concatenating simpler automata representing the set of possible tagging for each of the word composing the text. Figures 1(a)-(b) show these automata for the words *this* and *limit*. The three paths of the automaton of Figure 1(a) account for the fact that *this* may be a singular (sing) determiner (Det) or pronoun (Pro), or an adverb (Adv) as in: *Tom is this tall*. Similarly, the automaton of Figure 1(b) has different paths corresponding to the case where the word *limit* is a singular (sing) noun (N), or the infinitive (inf), imperative (imp), or present (pres) form of a verb (V), with the third (3rd) person singular (sing) form excluded in the latter case.

Simple observations can help derive a set of forbidden sequences represented by the automaton A of Figure 2. For example, when *this* is an adverb, it cannot be followed by a noun or a verb, and similarly, when it is a determiner, it cannot precede a verb unless the verb is a past or present participle. The automaton A can help reduce the ambiguities of the text T since it enforces that the sequences accepted must be in $L(T) \cap \overline{\Sigma^* L(A) \Sigma^*}$. Figure 3 shows the automaton of accepted sequences resulting from the application of the local grammar A to T.

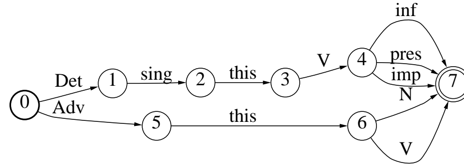

FIGURE 2 Finite automaton A representing a set of forbidden sequences.

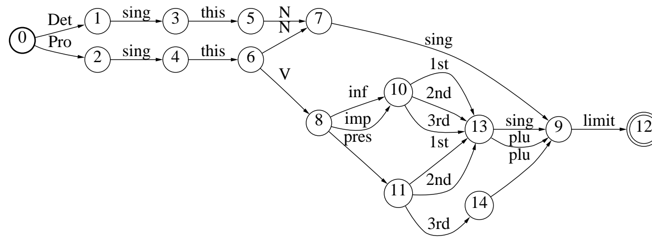

FIGURE 3 Accepted sequences $L(T) \cap \overline{\Sigma^* L(A) \Sigma^*}$.

The main problem for the application of a large local grammar A to a text automaton T is the efficient computation of an automaton representing $L(T) \cap \overline{\Sigma^* L(A) \Sigma^*}$. Complex local grammars automata may have in the order of several million transitions. The alphabet includes the vocabulary of the language considered, which, in English, has more than 200,000 words.

An automaton accepting $\Sigma^* L(A) \Sigma^*$ can thus be very large. Taking the complement of that automaton may lead to an even larger automaton since the worst case complexity of complementation is exponential for a non-deterministic automaton, and the result would yet need to be intersected with T.

The next section examines several algorithms for the computation of an automaton representing $L(T) \cap \overline{\Sigma^* L(A) \Sigma^*}$.

9.2 Algorithms

This section presents two local grammar algorithms. It first discusses the properties of a simple algorithm that can be viewed as the counter-part for local grammar algorithms of the straightforward quadratic-time string-matching algorithm and illustrates its application. A more efficient algorithm is then described in detail, including its pseudocode, and its optimization. In

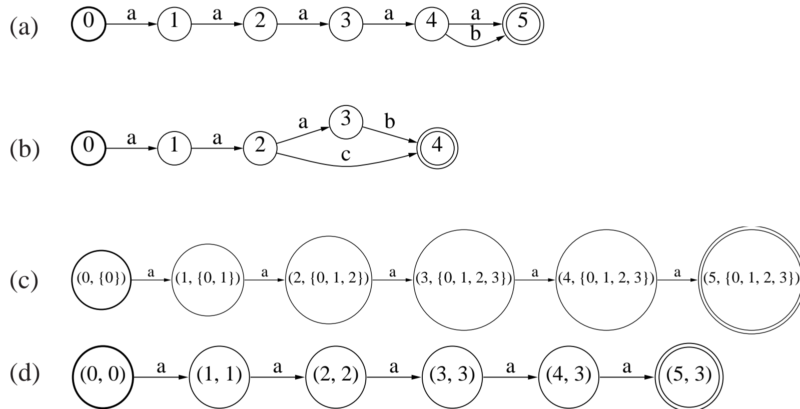

FIGURE 4 (a) Simple text automaton T_0. (b) Simple local grammar automaton A_0. (c) Result of the application of A_0 to T_0 using the simple algorithm. (d) Application of A_0 to T_0 using a more efficient algorithm.

what follows, the local grammar automaton A and the text automaton T will be assumed to be deterministic.

9.2.1 A simple algorithm

The problem of the application of a local grammar can be viewed as a generalization to automata of pattern-matching in text. A simple algorithm for the application of A to T is to search for all sequences accepted by A starting from each state of T. If a forbidden sequence is found, the appropriate transition is removed to disallow that sequence. This can be done by:

- simulating the presence of a self-loop labeled with all elements of Σ at the initial state of A;
- reading the paths of T starting from its initial state while pairing each state reached by a string x with the set of all states of A that can be reached by x from its initial state.

This describes the algorithm of Roche (1992). Figure 4(c) shows its result when using the simple text automaton of Figure 4(a) and the local grammar A_0 shown in Figure 4(b). Each state of the output automaton is a pair (p, s) where p is a state of T and s an element of the powerset of the states of A. At each state, the transitions of state p and those of the set of states in s are matched to form new transitions. In general, this operation may be very costly because of the large number of transitions leaving the states of s. Note that the transition labeled with b from the state $(4, \{0, 1, 2, 3\})$ to $(5, \{4\})$ is not constructed to disallow the forbidden sequence $aaab$ (state 4 is a final state of

A_0).

As it is clear from this example, the algorithm is very similar to the simple quadratic-time string-matching algorithm seeking to match a pattern at each position of the text, ignoring the information derived from matching attempts at previous positions.

The next section describes an algorithm that precisely exploits such information as with the linear-time string-matching algorithm of Knuth et al. (1977). Figure 4(d) shows the result of the application of that algorithm. Each state of the output automaton is identified with a pair of states (p, q) where p is a state of T and q the state of A corresponding to the longest (proper) suffix of the strings leading to p.

9.2.2 A more efficient local grammar algorithm

The application of a local grammar is directly related to the computation of a deterministic automaton representing $\Sigma^* L(A)$. Let A' be the automaton constructed by augmenting A with a self-loop at its initial state labeled with all elements of the alphabet Σ, and let $B = det(A')$ be the result of the determinization of A'. B recognizes the language $\Sigma^* L(A)$. To apply the local grammar A to T, we can proceed as for computing the intersection $B \cap T$, barring the creation of transitions leading to a state identified with a pair (p, q) where q is a final state of B.

In fact, since determinization can be computed on-the-fly (Aho et al., 1986, Mohri, 1997a), the full determinization of A' is not needed, only the part relevant to the computation of the intersection with T. However, if one wishes to apply the grammar to many different texts, it is preferable to compute B' once beforehand. In general, the computation of B' may be very costly though, in particular because of the alphabet size $|\Sigma|$ which can reach several hundred thousand.

There exists an algorithm for constructing a compact representation of the deterministic automaton representing $\Sigma^* L(A)$ using failure transitions (Mohri, 1997b). A failure transition is a special transition that is taken when no standard transition with the desired input label is found.

The algorithm can be viewed as a generalization to an arbitrary deterministic automaton A of the classical algorithms of Knuth et al. (1977) and that of Aho and Corasick (1975) that were designed only for strings or trees. When A is a tree, the complexity of the algorithm of Mohri (1997b) coincides with that of Aho and Corasick (1975): it is linear in the sum of the lengths of the strings accepted by A.

The following is the pseudocode of that algorithm in the case where A is acyclic.

LocalGrammar(A)

1 $E \leftarrow E \cup \{(i, \phi, i)\}$
2 ENQUEUE(S, i)
3 **while** $S \neq \emptyset$ **do**
4 $\quad p \leftarrow$ DEQUEUE(S)
5 $\quad$ **for** $e \in E[p]$ **do**
6 $\quad\quad q \leftarrow \delta(p, \phi)$
7 $\quad\quad$ **while** $q \neq i$ **and** $\delta(q, l[e]) =$ UNDEFINED **do** $q \leftarrow \delta(p, \phi)$
8 $\quad\quad$ **if** $p \neq i$ **and** $\delta(q, l[e]) \neq$ UNDEFINED
9 $\quad\quad\quad$ **then** $q \leftarrow \delta(q, l[e])$
10 $\quad\quad$ **if** $\delta(n[e], \phi) =$ UNDEFINED
11 $\quad\quad\quad$ **then** $\delta(n[e], \phi) \leftarrow q$
12 $\quad\quad\quad\quad$ **if** $q \in F$ **then** $F \leftarrow F \cup \{n[e]\}$
13 $\quad\quad\quad\quad L[n[e]] = L[n[e]] \cup \{n[e]\}$
14 $\quad\quad\quad\quad$ ENQUEUE$(S, n[e])$
15 $\quad\quad$ **else if** there exists $r \in L[old[n[e]]]$ such that $(r, \phi, q) \in E$
16 $\quad\quad\quad$ **then** $n[e] \leftarrow r$
17 $\quad\quad$ **else if** $old[q] \neq n[e]$
18 $\quad\quad\quad$ **then** create new state r
19 $\quad\quad\quad\quad$ **for** $e' \in E[n[e]]$ such that $l[e'] \neq \phi$ **do**
20 $\quad\quad\quad\quad\quad E \leftarrow E \cup \{(r, l[e'], old[n[e']])\}$
21 $\quad\quad\quad\quad E \leftarrow E \cup (r, \phi, q)$
22 $\quad\quad\quad\quad old[r] \leftarrow old[n[e]]$
23 $\quad\quad\quad\quad$ **if** $old[n[e]] \in F$ **then** $F \leftarrow F \cup \{r\}$
24 $\quad\quad\quad\quad L[old[n[e]]] = L[old[n[e]]] \cup \{r\}$
25 $\quad\quad\quad\quad n[e] \leftarrow r$
26 $\quad\quad\quad\quad$ ENQUEUE(S, r)
27 $\quad\quad$ **else** $n[e] \leftarrow q$

The algorithm takes as input a deterministic automaton A that it modifies to construct the desired local grammar automaton. States of A are visited in the order of a breadth-first search using a FIFO queue S. Each state q admits a failure transition labeled with ϕ. The destination state of that transition is the *failure state* of q, which is defined as the state reached by the longest proper suffix of the strings reaching q that are prefix of $L(A)$. Two distinct paths reaching q may correspond to two distinct failure states for q. In that case, q must be duplicated. Thus, the algorithm maintains the two following attributes: $old[q]$, the original state from which q was copied and, if q was originally in A (*i.e.* $old[q] = q$), $L[q]$, the list of the states obtained by copying q.

The outgoing transitions e of each state p extracted from the queue S (line 4) are examined. The candidate failure state q of $n[e]$ is determined (lines 6-

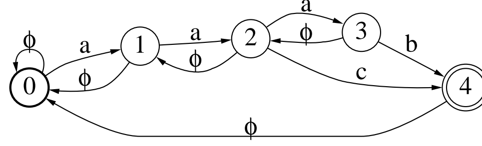

FIGURE 5 Finite automaton B'_0 recognizing $\Sigma^* L(A_0)$, where A_0 is the automaton of Figure 4. Failure transitions are marked with ϕ.

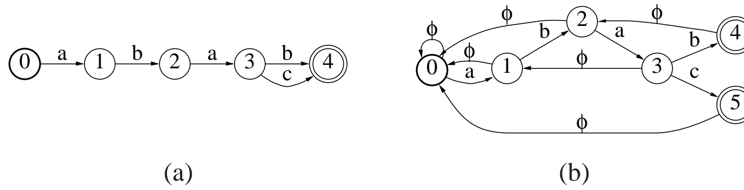

FIGURE 6 (a) Deterministic automaton A; (b) deterministic automaton B recognizing $\Sigma^* L(A)$. Transitions labeled with ϕ represent failure transitions.

10) as the first state on the failure path of p that has an outgoing transition labeled by $l[e]$. If $n[e]$ is not already assigned a failure state, its failure state is set to q and $n[e]$ is added to the queue (lines 10-14). If there exists a state r that has the same original state as $n[e]$ and has q as a failure state, then the destination of e is changed to r (lines 15-16). If q is not a copy of $n[e]$, then a new state r is created by copying $n[e]$, the failure state of r is set to q, the destination state of e is changed to r and r is added to the queue (lines 17-26). Otherwise, the destination state of e is changed to q (line 27).

When A is not acyclic, the condition of the test of line 17 needs to be generalized as described in detail in (Mohri, 1997b). An efficient implementation of this algorithm has been incorporated in the GRM library (Allauzen et al., 2005) with the command-line utility `grmlocalgrammar`.

Figure 5 shows the output of the algorithm when applied to the automaton A_0 of Figure 4(b). Each state admits a failure transition. The failure transition at the initial state is a self-loop. In such cases, the search for a default state can stop, e.g., at state 0, if a desired label such as b cannot be found, no further default state is considered. The automaton of Figure 5 is intersected with T_0 in the way previously described to produce the result (Figure 4(d)).

Figure 6(b) shows another illustration of the application where it is applied to the automaton of Figure 6(a). The special symbol ϕ is used to mark failure

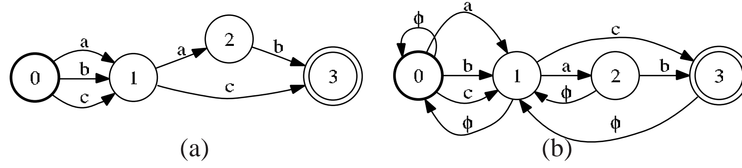

FIGURE 7 (a) Local grammar automaton A; (b) deterministic automaton B recognizing $\Sigma^* L(A)$ represented with failure transitions.

transitions.

The algorithm just described admits an on-the-fly implementation which makes it suitable for expanding only those states and transitions of the result need for the intersection with T.

An offline construction is preferable when multiple applications of the local grammar are expected. Unlike the algorithm presented in the previous section, the determinization of A' is then computed just once. The resulting automaton B' is compact thanks to the use of failure transitions.

The use of B' can be further optimized in a way similar to what can be done in the case of the algorithm of Knuth et al. (1977) using the following observation: if a label a is unavailable at q, it is also unavailable at the default state q' of q if the set of labels at q' is included in set of labels at q. Let the *context of* q be defined by:

$$C(q) = \{a \in \Sigma : \delta(q, a) \neq \emptyset\}.$$

To speed up the use of default transitions, the new transition function δ' can thus be defined as follows:

$$\delta'(q, \phi) \leftarrow \begin{cases} \delta(q, \phi) & \text{if } C(\delta(q, \phi)) \not\subseteq C(q) \text{ or } \delta(q, \phi) = q; \\ \delta'(\delta(q, \phi), \phi) & \text{otherwise.} \end{cases}$$

For example, the context of state 3 contains that of its default state 1 in the automaton of Figure 6(b). Thus, its default transition can be redefined to point to the default state of state 1, that is state 0.

Figures 7 and 8 provide a full example of application of a local grammar using the algorithm described. Figure 7(a) shows an example of a local grammar automaton A. The application of the algorithm produces the compact deterministic automaton B of Figure 7(b) represented with failure transitions.

Figure 8(a) shows a text automaton and Figure 8(b) the result of the application of the application of A to T obtained by intersecting B with T. The dotted transition is a transition not constructed during that intersection since it leads to the state pair $(2, 3)$ where 3 is a final state of B.

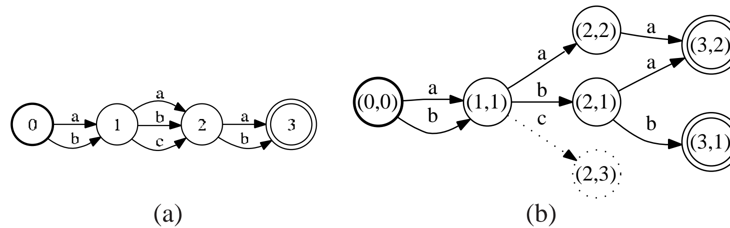

FIGURE 8 (a) Text automaton T; (b) Result of the application of the local grammar A to T.

9.3 Conclusion

Accurate local grammar automata are useful tools for disambiguation. They can significantly speed up the application of further text processing steps such as part-of-speech tagging or parsing. We gave a brief overview of several local grammar algorithms, including an efficient algorithm for their application to a text represented by an automaton.

Another natural way to define local grammars is to use context-dependent rewrite rules. Context-dependent rules can be efficiently compiled into finite-state transducers that can then be readily applied to an input text automaton (Kaplan and Kay, 1994, Mohri and Sproat, 1996). They can be further generalized to weighted context-dependent rules compiled into weighted transducers (Mohri and Sproat, 1996).

References

Aho, Alfred V. and Margaret J. Corasick. 1975. Efficient string matching: An aid to bibliographic search. *Communication of the Association for Computing Machinery* 18 (6):333–340.

Aho, Alfred V., Ravi Sethi, and Jeffrey D. Ullman. 1986. *Compilers, Principles, Techniques and Tools*. Addison Wesley: Reading, MA.

Allauzen, Cyril, Mehryar Mohri, and Brian Roark. 2005. The Design Principles and Algorithms of a Weighted Grammar Library. *International Journal of Foundations of Computer Science* (to appear).

Gross, Maurice. 1968. *Grammaire transformationnelle du francais.*, vol. 1, Syntaxe du verbe. Larousse.

Gross, Maurice. 1997. The Construction of Local Grammars. In *Finite-State Language Processing*, pages 329–354. The MIT Press, Cambridge, Massachusetts.

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

Knuth, D.E., J.H. Morris, and V.R. Pratt. 1977. Fast Pattern Matching in Strings. *SIAM Journal on Computing* 6:323–350.

Koskenniemi, Kimmo. 1990. Finite-State Parsing and Disambiguation. In *Proceedings of the thirteenth International Conference on Computational Linguistics (COLING'90), Helsinki, Finland*.

Maurel, Denis. 1989. *Reconnaissance de séquences de mots par automates. Adverbes de date*. Ph.D. thesis, Université Paris 7.

Mohri, Mehryar. 1997a. Finite-State Transducers in Language and Speech Processing. *Computational Linguistics* 23:2.

Mohri, Mehryar. 1997b. String-Matching with Automata. *Nordic Journal of Computing* 4:2.

Mohri, Mehryar and Richard Sproat. 1996. An Efficient Compiler for Weighted Rewrite Rules. In 34*th Meeting of the Association for Computational Linguistics (ACL '96), Proceedings of the Conference, Santa Cruz, California*.

Roche, Emmanuel. 1992. Text disambiguation by finite state automata, an algorithm and experiments on corpora. In *Proceedings of COLING-92*.