Locally Tree-shaped Sentence Automata and Resolution of Ambiguity

Jussi Piitulainen
Research Unit for Multilingual Language Technology
Department of General Linguistics
University of Helsinki
jpiitula@ling.helsinki.fi

Abstract

The framework of finite state intersection grammars is directed towards practical representation and parsing of running text. A problem in parsing has been that intersection with successive rule automata can produce prohibitively large intermediate sentence automata. See (Koskenniemi, Tapanainen and Voutilainen 1992) and (Koskenniemi 1990) and (Voutilainen and Tapanainen 1993).

This paper sketches a data structure and parsing method that may help to compute the intersection of a sentence automaton with all rule automata while keeping the size of intermediate sentence automata under control.

1 Representation of sentence readings

Sentence readings are finite sequences of word forms, morphosyntactic labels and boundary labels (Voutilainen 1994, Voutilainen and Tapanainen 1993). The structure of a sentence reading is such that each word form reading contains the word form, a base form, a few morphological labels and one or two syntactic labels, and word form readings are separated by boundary labels. For example, the correct reading of the sentence 'And time began seriously to pass.' in figure 1 identifies 'time' as a subject @SUBJ, and 'began' as a main verb in a main clause @MV MAINC@, and so on. Other readings might misidentify 'time' as a verb or 'to' as a preposition, or correctly identify 'time' as a noun but misidentify it syntactically as an object or an apposition. An incorrect choice for a word boundary label would indicate the presence of more than one finite verb form.

There are five alternative word boundary labels: @@ begins and ends a sentence, embedded finite clauses are enclosed between @< and @>, a boundary where a finite clause ends and a new one starts is labeled with @/, and @ is used for others.

A finite set of readings is represented as an acyclic finite state automaton called a sentence automaton. Word forms in the sentence have all lexically possible morphosyntactic readings as alternatives, and each word boundary is made four or sometimes five ways ambiguous. The initial number of readings in a sentence automaton is the product of the lexical and word boundary ambiguities in the sentence.

Proceedings of NODALIDA 1995, pages 50-58

```
                                                    @@
<and> and CC                    @CC                 @
<time> time N NOM SG            @SUBJ               @
<began> begin V PAST VFIN       @MV MAINC@          @
<seriously> serious ADV         @ADVL               @
<to> to INFMARK                 @aux                @
<pass> pass V INF               @mv OBJ@            @
<.>                             @fullstop           @@
```

Figure 1 A sentence reading

The grammar is represented as a number of finite state automata called rule automata. Each rule automaton is constructed to accept all readings that are to be grammatical and to reject some readings that are to be ungrammatical. In other words, the grammar is taken to be the intersection of individual rules.

Sentence automata will be drawn with the the start state on the left, the direction of edges from left to right, and the final state as a double circle on the right.

2 The problem

The task is to compute the intersection $S \cap R_0 \cap \cdots \cap R_{n-1}$ of a sentence automaton S with all rule automata $R_0, \ldots, R_{n-1}$. The main practical problem is that sometimes intersection with successive rule automata produces prohibitively big intermediate sentence automata. Any method that computes the same final result can be used instead of the straightforward intersection; this paper proposes one such method.

The number of readings represented by the sentence automaton decreases monotonically during the parsing process. The number of states in the sentence automaton tends to increase at first; the initial automaton can be very compact precisely because it contains all alternatives, and a smaller set may be represented by a bigger automaton.

The removal of the dashed edges in figure 2 leaves two minimal automata of which the bigger represents the smaller set of readings. This shows how removal of readings can require addition of states and edges.

The two automata in figure 2 represent a set of four readings. The upper automaton is minimal and the lower automaton is (almost) maximal. Consider now a rule that rejects the sequence $\langle x, a, y, a, z \rangle$ and accepts the other three. No edge can be removed from the minimal automaton since every edge belongs to one of $\langle x, a, y, b, z \rangle$ and $\langle x, b, y, a, z \rangle$ and $\langle x, b, y, b, z \rangle$. In the maximal automaton, the dashed edges and the state between them belong only to the rejected sequence and can be removed.

The final sentence automaton $S \cap R_0 \cap \cdots \cap R_{n-1}$ represents the set of correct readings for the sentence. If the grammar is accurate, this automaton is again quite small because it represents a very small set of readings.

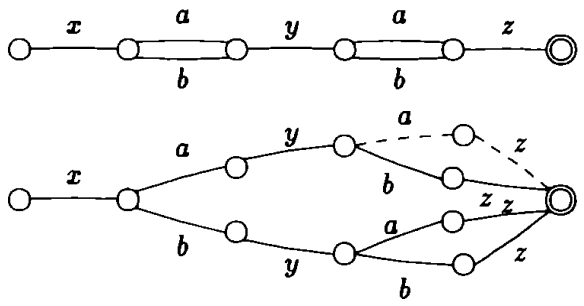

Figure 2 A minimal automaton and a maximal automaton

3 New representation of sentence automata

This section introduces a new representation for sentence automata. The new data structure is designed to facilitate separation of two operations that intersection would combine, namely removal of readings and expansion of the sentence automaton. This separation should help keep the size of intermediate automata under control.

Edges and states can be removed if they are not parts of any reading that should remain. In a tree-shaped automaton, all removable readings can be removed by removing such edges and states. Since it is strictly impractical to make the whole sentence automaton tree-shaped, the automaton is made tree-shaped only locally. Initially, the local trees span readings of word forms, but they can be expanded to span longer intervals.

There must be states with more than one immediate predecessor. These are immediate successors of the leaves of local trees. Since the trees initially span word forms, the word boundaries correspond to collections of the states that may have several immediate predecessors. These collections will be called slices.

Each slice of a sentence automaton contains exactly one state of every path between the start state and the final state. The first slice contains the state that immediately follows the start state, and the last slice contains the final state. Initially slices contain only one state.

The two automata in figure 3 represent a set of a few readings of the sentence 'Time passed'. Punctuation and some labels, notably **MAINC@**, are omitted to save space. The upper automaton is locally tree-shaped; the lower automaton is an ordinary minimal automaton.

The rectangles in the upper automaton represent the three slices corresponding to the three word boundaries in the sentence.

Each state in a slice is a root of a local tree. The leaves of the tree are immediate predecessors of some roots in the next slice. Note that a state that is properly between slices has only one immediate predecessor, so that the trees do not share structure.

The data structure representing a sentence automaton should give easy access to the information needed in the algorithms to be described, in addition to the underlying automaton structure. First, the containing slice, if any, should be available given a state. Second, the previous state or states should be available given a

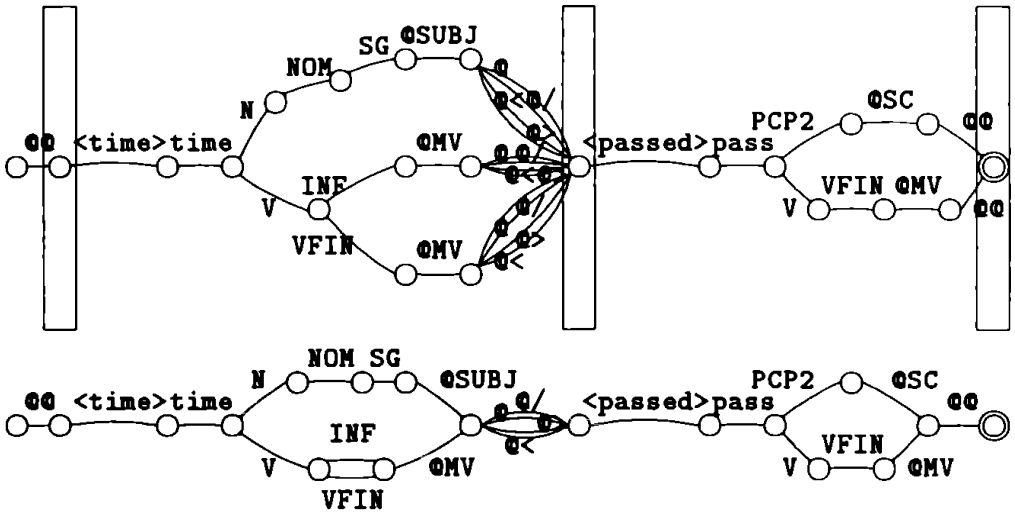

Figure 3 Locally tree-shaped vs. minimal

state. Third, the states in slices and immediate predecessors of states in slices will be assigned sets of rule states that should be available given such a state. It should be easy to iterate over all edges that leave a state.

4 Operations on the new data structure

The new data structure is designed so that removal and addition of states and edges are separate operations. This is in contrast to intersection that not only removes readings but also expands the sentence automaton in a way that is difficult to control.

In contrast to sentence automata, next to nothing is assumed about the representation of rule automata. All that is required is that it is possible to compute the state where a given state takes a given symbol, and to decide whether there is any way to reach a final state from a given state.

A state is called a sink if there is no way to reach a final state from it. A rule can fail by going into a sink long before reaching the end of an input sequence.

4.1 Initialization and propagation

In the new representation, sentence automata contain two special kinds of states, namely the roots of the local trees and the leaves of the local trees. The roots are in slices, and each leaf is an immediate predecessor of a root. (The start state is also thought of as a leaf.) Readings are removed in two steps called propagation and removal.

The propagation step assigns sets of states of a rule automaton to the roots and leaves of the local trees. Initially, a set containing the start state of the rule automaton is assigned to the start state of the sentence automaton. There is only one path from the start state of the sentence automaton to the first root, the one labeled @@; the set containing the state to which the start state of the rule automaton takes the label @@ is assigned to the first root.

Once initialized, the propagation step continues by assignment of sets of rule states to the roots and leaves of each local tree in turn. For example, a certain rule

automaton goes to a sink state when it encounters the label @MV after it has already encountered @MV but not yet encountered a clause boundary. Let state 1 be the start state, state 2 be the state after the first @MV and state 5 be the sink state. In the upper automaton in figure 4, the root has been reached through paths that contain one @MV and paths that do not contain @MV. Therefore, the root is assigned the set of two rule states $\{1,2\}$. When this set is propagated further, the rule automaton stays in these two states until it encounters the @MV label; then the state set would be $\{2,5\}$ but the sink state 5 is simply ignored.

In the lower automaton in figure 4, the root has been reached only through paths that contain one @MV. The state set after the second @MV contains only the sink state and becomes effectively empty.

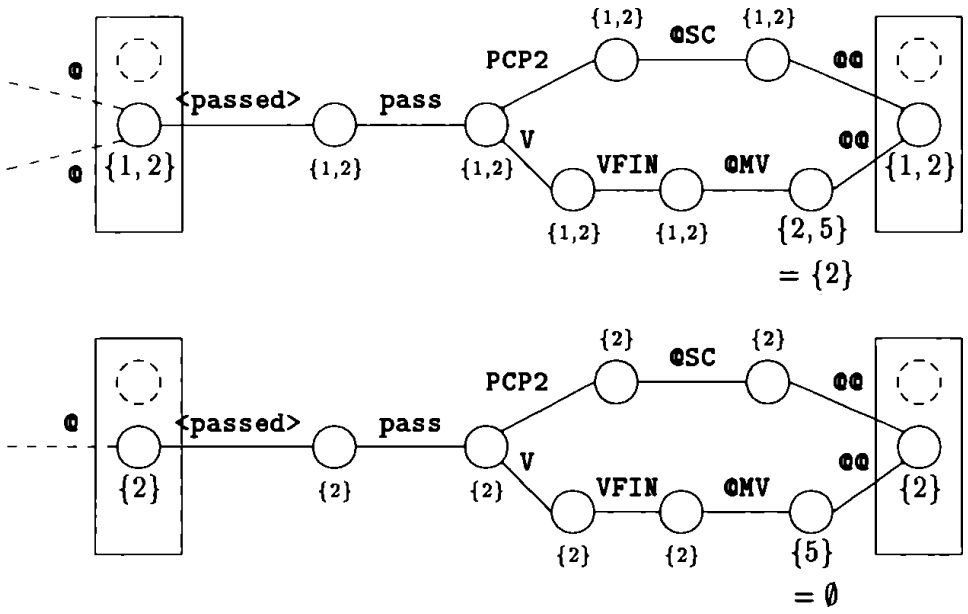

Figure 4 Propagation of a rule state set

The first root has been assigned in the initialization step. There is a unique path from the root of a local tree to a leaf of that tree. This path and the rule states in the root determine the set of rule states that are assigned to the leaf. Rule states in the leaf are those to which some rule state in the root takes the path from the root to the leaf, excluding sinks. When all the rule states would be sinks, this set is empty.

Unlike leaves, the roots in a slice can have several predecessors. A root is assigned the set of rule states that are reachable from some rule state in some of its predecessors by some path from the leaf to the root.

4.2 Removing paths that block propagation

The sets of rule states in the leaves help to remove states and edges. First, if a leaf is assigned the empty set of rule states, there is no way for the rule to reach this leaf. The leaf can be removed, and edges and states preceding it can be removed back to the first state with more than one immediate successor. All boundary edges forward can be removed, and if the following root had no other immediate

predecessor, removal can continue forward. This means that the edges labeled V VFIN @MV @@ in the lower automaton in figure 4 can be removed.

Second, if all the rule states in some leaf take some boundary edge to a sink, the edge can be removed. If all the edges from the leaf to a root are removable this way, removal can continue backward and forward.

Third, a non-final state in the last slice can be treated as if it were a sink. There is no way to reach another final state from the last state in the sentence automaton.

Only such readings are removed as would be removed by intersection. However, not all such readings can be removed this way. To allow removal of further readings, the sentence automaton may need to be expanded.

4.3 Expanding the sentence automaton locally

In the new method, the sentence automaton cannot grow while readings are removed. A separate operation of local expansion is provided to open up removal opportunities.

Observe that the initial sentence automaton is tree-shaped between any pair of successive slices. The expansion makes it tree-shaped (rather forest-shaped) between some given pair of slices. Each state that is properly between the two slices will have exactly one immediate predecessor.

The example sentence 'Time passed' is so short that there is only one way to choose a pair of slices for expansion. The initial automaton is shown in figure 3; the expanded automaton is shown in figure 5. For longer sentences, such maximal expansion is not feasible.

The first word form 'time' is three ways ambiguous and the word boundary after it is four ways ambiguous; together 'time' and the boundary are twelve ways ambiguous. Expansion makes twelve copies of the state in the slice between the readings of 'time' and 'passed', and the tree of 'passed' is turned into a forest of twelve identical trees.

Much of the new structure is easy to remove. Particularly interesting are the three trees of 'passed' that follow the boundary symbol @/. This boundary symbol separates finite clauses: there should be the label VFIN both before and after it. This constraint is not enough to remove anything in the automaton of figure 3 since VFIN occurs in one of the readings of 'time', but in the automaton of figure 5, two of the three trees can be removed. For example, after @@ <time> time N NOM SG @SUBJ @/ the rule should be in a sink state.

Another impossibility is to have @< and @@ or @< and @< as successive boundaries. The former constraint would allow removal of much of the structure in the expanded automaton: the label @@ would take a rule to a sink.

Parts of the lower four trees of 'passed' are removable by the constraint that the boundary symbol between two finite verb forms VFIN can not be @, so that the second VFIN would take a rule to a sink.

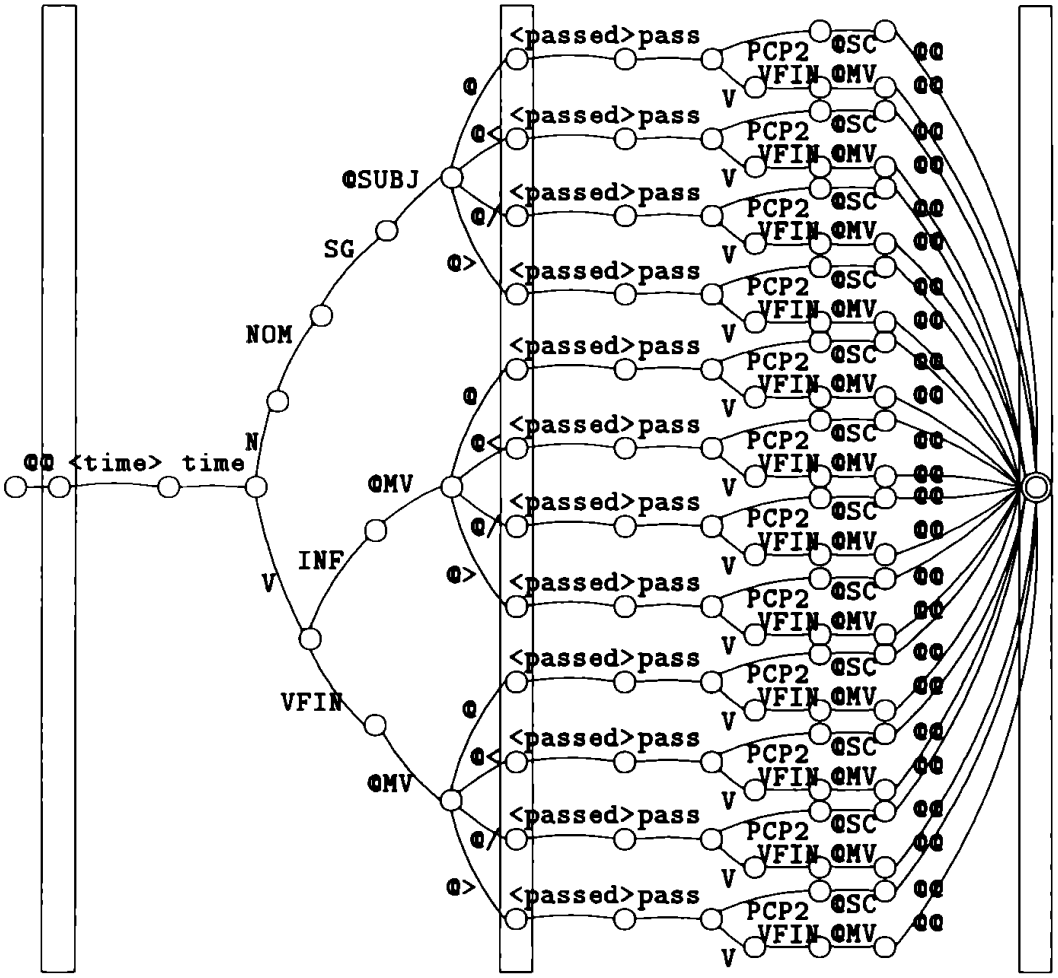

Figure 5 An expanded sentence automaton

5 Computation strategies

The high level strategy to use is an open question. The operations of propagation, removal and expansion can be used in different orders. In this method, rules need to be used more than once, and the order of expansions and removals may affect efficiency significantly.

The simple minded but prohibitively expensive strategy would be to first maximize the sentence automaton and then just use every rule in turn.

Another strategy would be to exercise all rule automata on a short interval in the beginning of the sentece automaton first, and then lengthen the interval. A third strategy would be to expand short intervals and then propagate and remove.

More intelligent strategies might take into account special properties of each sentence automaton, or special properties of some rules. Or such intervals might be expanded that contain relatively few readings.

Kimmo Koskenniemi (personal communication) suggested first propagating a rule into the sentence automaton and then studying the sets of rule states to see whether this rule can allow removals after expansion. In the positive case, expand and then propagate again.

6 Some details

This section gives a more detailed description of the operations on the locally tree-shaped automata than was given before. All automata here are deterministic finite state automata.

An automaton is a finite set of uniquely labeled states, partitioned into final and non-final states, with one state designated as the start state.

All states will be seen as total functions taking sequences to states, as follows. When s and t are states and a path from s to t is labeled by a sequence of alphabet symbols w, the notation $s\,w$ designates the state t. The set of all prefixes of such sequences w that s takes to a final state is called the domain of s. A state with empty domain is called a sink; if w is not in the domain of s, then $s\,w$ is a sink.

Each state s satisfies the equations $s\ \langle\rangle = s$ and $s\ \langle a, b, \ldots\rangle = (s\ \langle a\rangle)\ \langle b, \ldots\rangle$ for all $a, b, \ldots$ in the alphabet. In a minimal automaton, a sink s is a non-final state that satisfies $s\,w = s$ for all sequences w.

If $s\ \langle a\rangle = t$ for some symbol a, then s is said to be an immediate predecessor of t, and t is said to be an immediate successor of s.

In the process to be specified, a sentence automaton $\mathcal{S}$ is represented as a sequence of slices $(\mathcal{S}_0, \ldots, \mathcal{S}_{n-1})$. It must be replaced by successive sentence automata so that the final result represents the intersection of the original sentence automaton with a number of rule automata $R_0, \ldots, R_{k-1}$.

6.1 Propagation

Rule states can be propagated from $\mathcal{S}_i$ to $\mathcal{S}_{i+1}$. An intermediate step is to propagate them to the leaves of each local tree with root in $\mathcal{S}_i$. When a path from a root to a leaf is labeled w and the set of rule states assigned to the root is R, the set of rule states in the leaf will be $\{\, r\,w \mid r \in R \,\}$ excluding sinks.

After propagation to the leaves, an immediate predecessor p of a state t in $\mathcal{S}_{i+1}$ contains the correct set R' of rule states. If D is the domain of p, the set of rule states in t will be a superset of $\{\, r\ \langle d\rangle \mid r \in R' \text{ and } \langle d\rangle \in D \,\}$ excluding sinks. Other immediate predecessors of t produce other states.

Rule states can be propagated from the start state to each slice in turn. If the start state of a rule is r_0, the state in the first slice will contain just the state $r_0\ \langle \text{@@} \rangle$.

6.2 Removal

When rule states have been propagated from $\mathcal{S}_0$ to $\mathcal{S}_i$, it may be possible to remove edges and states that precede states in $\mathcal{S}_i$.

If p is an immediate predecessor of some state s in $\mathcal{S}_i$ and the set of rule states in p is empty, then the rule takes to a sink all sequences that label paths from the start state to p. The state p can be removed together with preceding edges and states back to the first state that has more than one edge in its domain.

If the set of rule states in p is not empty but there are several edges from p to s, all the rule states can take some of the edges to a sink. Such edges can be removed.

6.3 Expansion

When the sentence automaton is expanded between S_i and S_{i+2}, and some state s in S_{i+1} has k immediate predecessors, the state s and its local tree are replaced with k copies, each with its own immediate predecessor out of the immediate predecessors of the original s.

7 Summary

Intersection of finite state automata reduces readings but may add structure in the automata. Separation of the two effects may help to keep the size of a sentence automaton under control. Keeping the sentence automaton locally tree-shaped makes this separation possible.

Future work will involve implementation of the parser and empirical study of various prasing strategies.

Acknowledgements

Special thanks to Anssi Yli-Jyrä, Kari Granö, Atro Voutilainen and Kimmo Koskenniemi, and thanks to Timo Lahtinen, Timo Järvinen, Mikko Lounela and the two referees, for reading and comments.

The two example sentences are from the Douglas Adams novel, Dirk Gently's Holistic Detective Agency, first published in 1987 by William Heinemann Ltd.

References

Kimmo Koskenniemi. 1990. Finite-state Parsing and Disambiguation. In Hans Karlgren (ed). *COLING-90: Papers Presented to the 13th International Conference on Computational Linguistics, Vol. 2.* Helsinki, Finland.

Kimmo Koskenniemi, Pasi Tapanainen and Atro Voutilainen. 1992. Compiling and Using Finite-State Syntactic Rules. In *Proceedings of COLING-92, Vol. I.* Nantes, France.

Atro Voutilainen and Pasi Tapanainen. 1993. Ambiguity resolution in a reductionistic parser. In *Proceedings of EACL-93.* Utrecht, The Netherlands.

Atro Voutilainen. 1994. Designing a Parsing Grammar. In *Three Studies of Grammar-Based Surface Parsing of Unrestricted English Text.* Ph.D. dissertation. Helsinki, Finland.