Backward and Forward Bisimulation of Finite Tree Automata

J. Högberg, A. Maletti, and J. May

16 juli 2007

Outline

1. Background and motivation

2. Bisimulation minimisation of tree automata

3. Partition refinement algorithms

4. An NLP application

5. Work in progress

Ranked alphabets and trees

Ranked alphabets and trees

A ranked alphabet is a finite set of symbols

$$\Sigma = \bigcup_{k \in N} \Sigma_{(k)} \quad .$$

Ranked alphabets and trees

A ranked alphabet is a finite set of symbols

$$\Sigma = \bigcup_{k \in N} \Sigma_{(k)} \quad .$$

The symbols in $\Sigma_{(k)}$ have rank k.

Ranked alphabets and trees

A ranked alphabet is a finite set of symbols

$$\Sigma = \bigcup_{k \in N} \Sigma_{(k)} \quad .$$

The symbols in $\Sigma_{(k)}$ have rank k.

$$\Sigma = \Sigma_{(0)} \cup \Sigma_{(1)} \cup \ldots$$

$$\Sigma_{(0)} = \{1, 2\}, \quad \Sigma_{(1)} = \{-\},$$
$$\Sigma_{(2)} = \{+, \times\}, \; \Sigma_{(n)} = \emptyset \quad , n \geq 3$$

Ranked alphabets and trees

A ranked alphabet is a finite set of symbols

$$\Sigma = \bigcup_{k \in N} \Sigma_{(k)} \quad .$$

$$\Sigma = \Sigma_{(0)} \cup \Sigma_{(1)} \cup \ldots$$

$$\Sigma_{(0)} = \{1, 2\}, \quad \Sigma_{(1)} = \{-\},$$
$$\Sigma_{(2)} = \{+, \times\}, \; \Sigma_{(n)} = \emptyset \;\; , n \geq 3$$

The symbols in $\Sigma_{(k)}$ have rank k.

The set of trees over Σ is written T_{Σ}.

Ranked alphabets and trees

A ranked alphabet is a finite set of symbols

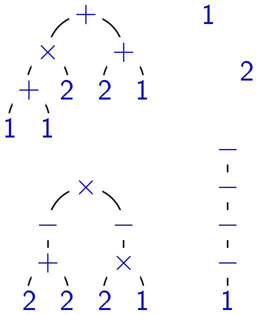
$$\Sigma = \bigcup_{k \in N} \Sigma_{(k)} \ .$$

The symbols in $\Sigma_{(k)}$ have rank k.

The set of trees over Σ is written T_Σ.

$$\Sigma = \Sigma_{(0)} \cup \Sigma_{(1)} \cup \ldots$$

$$\Sigma_{(0)} = \{1, 2\}, \quad \Sigma_{(1)} = \{-\},$$
$$\Sigma_{(2)} = \{+, \times\}, \ \Sigma_{(n)} = \emptyset \ \ , n \geq 3$$



Ranked alphabets and trees

A ranked alphabet is a finite set of symbols

$$\Sigma = \bigcup_{k \in N} \Sigma_{(k)} \quad .$$

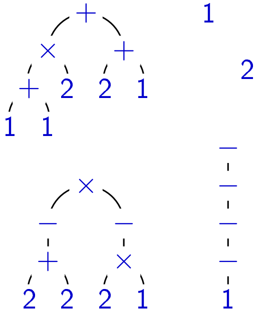
The symbols in $\Sigma_{(k)}$ have rank k.

The set of trees over Σ is written T_Σ.

A tree language w.r.t. Σ is simply a subset of T_Σ.

$$\Sigma = \Sigma_{(0)} \cup \Sigma_{(1)} \cup \dots$$

$$\Sigma_{(0)} = \{1, 2\}, \quad \Sigma_{(1)} = \{-\},$$
$$\Sigma_{(2)} = \{+, \times\}, \Sigma_{(n)} = \emptyset \quad , n \geq 3$$



Finite tree automata

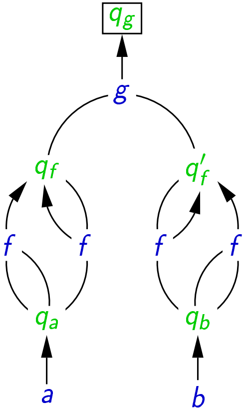
A finite tree automaton (fta) is a tuple (Q, Σ, δ, F) where

- Q is a finite set of states,
- Σ is a ranked input alphabet,
- δ is a finite set of transition rules in the form

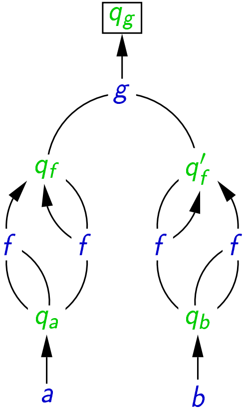
  $$f(q_1, \ldots, q_n) \to q_{n+1} \quad ,$$

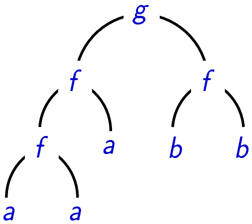
  where $f \in \Sigma_{(n)}$, and $q_1, \ldots, q_{n+1} \in Q$, for some $n \in \mathbb{N}$.
- Finally, $F \subseteq Q$ is a set of accepting states.

The language accepted by an fta

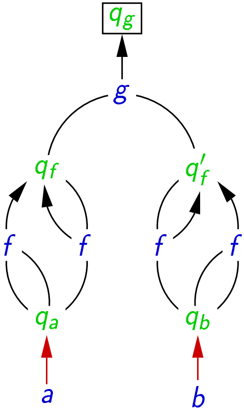


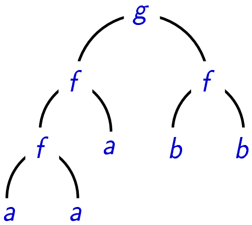
The language accepted by an fta



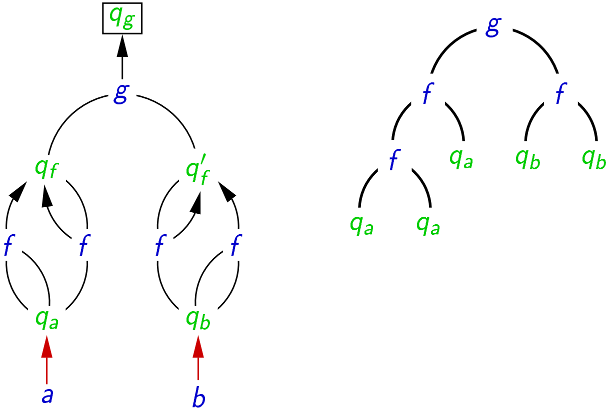


The language accepted by an fta

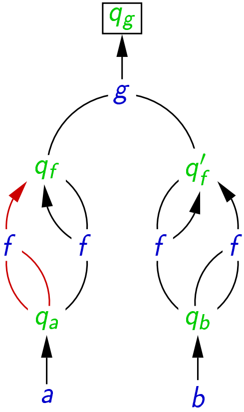


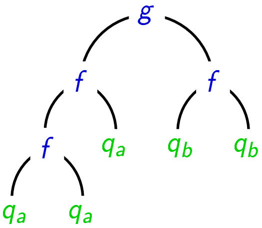


The language accepted by an fta

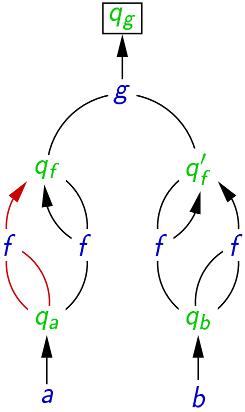


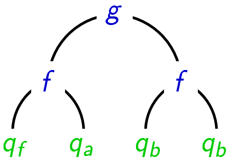
The language accepted by an fta



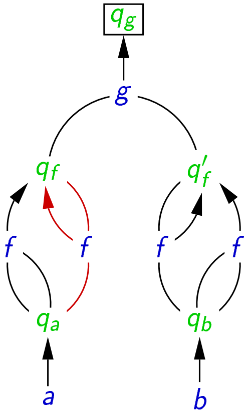


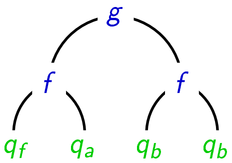
The language accepted by an fta



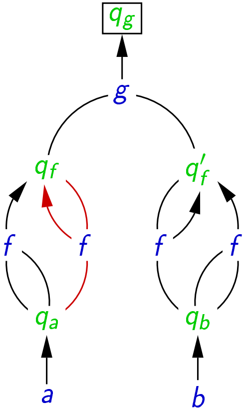


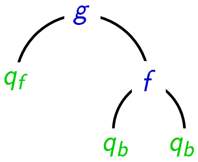
The language accepted by an fta



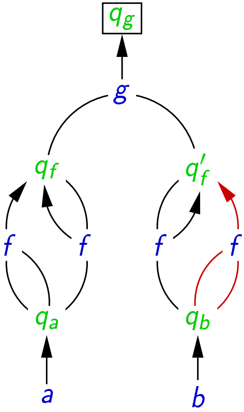


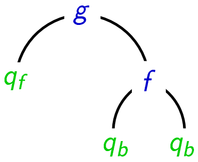
The language accepted by an fta



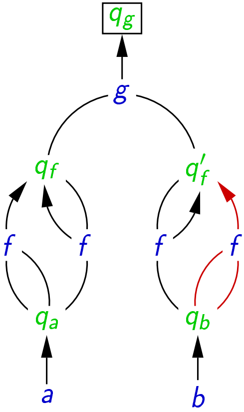


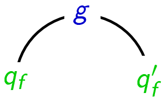
The language accepted by an fta



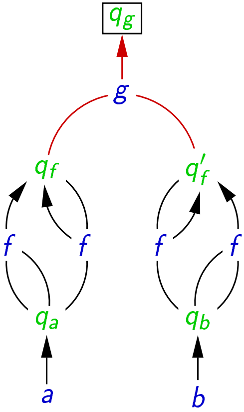


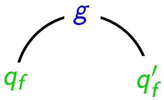
The language accepted by an fta



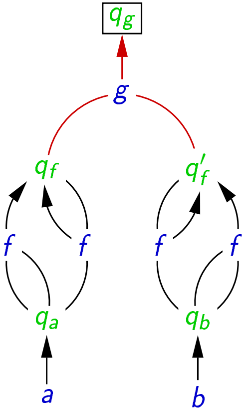


The language accepted by an fta





The language accepted by an fta



q_g

Applications

Finite tree automata . . .

- offer a nice combination of generative power and analytical transparency.
- are useful in areas such as lexical analysis, model checking and natural language processing.

To allow for efficient computations, we want to work with as small fta as possible. This makes a minimisation algorithm a useful tool.

Minimisation of tree automata

The Problem

Given an fta, find a minimal language equivalent fta.

Minimisation of tree automata

The Problem

Given an fta, find a minimal language equivalent fta.

The deterministic case is efficiently solvable, and the solution is always unique.

Minimisation of tree automata

The Problem

Given an fta, find a minimal language equivalent fta.

The deterministic case is efficiently solvable, and the solution is always unique.

The general case, however,

Minimisation of tree automata

The Problem

Given an fta, find a minimal language equivalent fta.

The deterministic case is efficiently solvable, and the solution is always unique.

The general case, however,

- lacks a unique solution,

Minimisation of tree automata

The Problem

Given an fta, find a minimal language equivalent fta.

The deterministic case is efficiently solvable, and the solution is always unique.

The general case, however,

- lacks a unique solution,
- is PSPACE complete [Meyer and Stockmeyer, 1972], and

Minimisation of tree automata

The Problem

Given an fta, find a minimal language equivalent fta.

The deterministic case is efficiently solvable, and the solution is always unique.

The general case, however,

- lacks a unique solution,
- is PSPACE complete [Meyer and Stockmeyer, 1972], and
- efficient approximation within a constant factor is not possible unless $P = NP$.

Minimisation of tree automata

The Problem

Given an fta, find a minimal language equivalent fta.

The deterministic case is efficiently solvable, and the solution is always unique.

The general case, however,

- lacks a unique solution,
- is PSPACE complete [Meyer and Stockmeyer, 1972], and
- efficient approximation within a constant factor is not possible unless $P = NP$.

Any efficient algorithm that searches for a solution to the general problem, must thus use heuristics.

Bisimulation

The notion of bisimularity is due to R. Milner.

Intuitively, two states are bisimilar if they serve the same purpose.

We adopt P. Buchholz definitions and extend these to trees:

- Backward bisimulation Two states are bisimilar if every tree that is mapped to the one state is also mapped to the other.
- Forward bisimulation Two states are bisimilar if they can always be exchanged for each other during a run on an input tree t, without affecting the way t is classified.

Backward bisimulation

Let $A = (Q, \Sigma, \delta, F)$ be an fta. An equivalence relation $\simeq$ on Q is a backward bisimulation if $p \simeq q$ means that

$$f(p_1, p_2, \ldots, p_k) \to p \quad ,$$

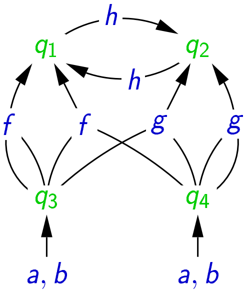
implies that there exists a rule

$$f(q_1, q_2, \ldots, q_k) \to q \quad ,$$

such that $p_i \simeq q_i$, for every $i \in \{1, \ldots, k\}$, and vice versa.

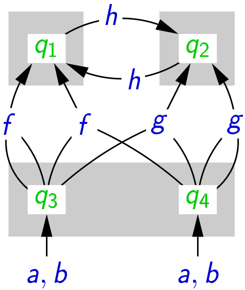
Minimisation w.r.t. backward bisimulation

Consider a backward bisimulation on the state space of the automaton below. By collapsing the states in each equivalence class into a single state, we obtain a smaller, language equivalent automaton.



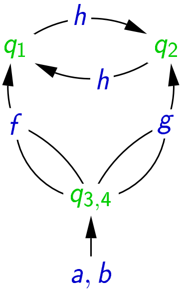
Minimisation w.r.t. backward bisimulation

Consider a backward bisimulation on the state space of the automaton below. By collapsing the states in each equivalence class into a single state, we obtain a smaller, language equivalent automaton.



Minimisation w.r.t. backward bisimulation

Consider a backward bisimulation on the state space of the automaton below. By collapsing the states in each equivalence class into a single state, we obtain a smaller, language equivalent automaton.



Forward bisimulation

Let $A = (Q, \Sigma, \delta, F)$ be an fta. An equivalence relation $\simeq$ on Q is a forward bisimulation if $p \simeq q$ means that

- $q \in F$ if and only if $q \in F$, and
- the fact that

  $$f(p_1, \ldots, p_{i-1}, p, p_i, \ldots, p_k) \to p_{k+1} \quad ,$$

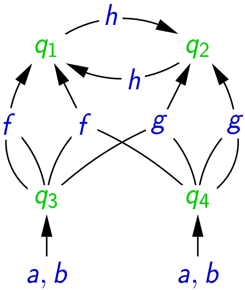
  where $i \in \{1, \ldots, k\}$, implies that there exists a rule

  $$f(p_1, \ldots, p_{i-1}, q, p_i, \ldots, p_k) \to q_{k+1} \quad ,$$

  such that $q_{k+1} \simeq q_{k+1}$, and vice versa.

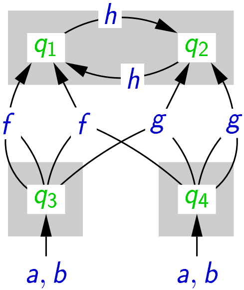
Minimisation w.r.t. forward bisimulation

Consider a forward bisimulation on the state space of the automaton below. By collapsing the states in each equivalence class into a single state, we obtain a smaller, language equivalent automaton.



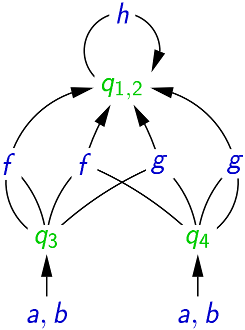
Minimisation w.r.t. forward bisimulation

Consider a forward bisimulation on the state space of the automaton below. By collapsing the states in each equivalence class into a single state, we obtain a smaller, language equivalent automaton.



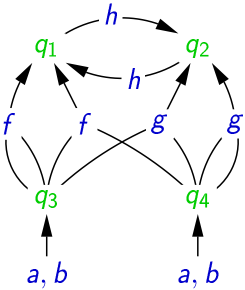
Minimisation w.r.t. forward bisimulation

Consider a forward bisimulation on the state space of the automaton below. By collapsing the states in each equivalence class into a single state, we obtain a smaller, language equivalent automaton.



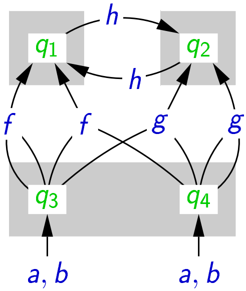
Combining the two types of minimisation

In a real-life application, one might wish to combine the two types of bisimulation to obtain even smaller output automata.



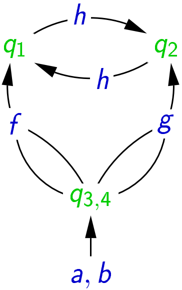
Combining the two types of minimisation

In a real-life application, one might wish to combine the two types of bisimulation to obtain even smaller output automata.



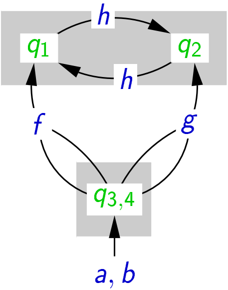
Combining the two types of minimisation

In a real-life application, one might wish to combine the two types of bisimulation to obtain even smaller output automata.



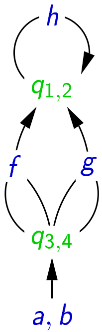
Combining the two types of minimisation

In a real-life application, one might wish to combine the two types of bisimulation to obtain even smaller output automata.

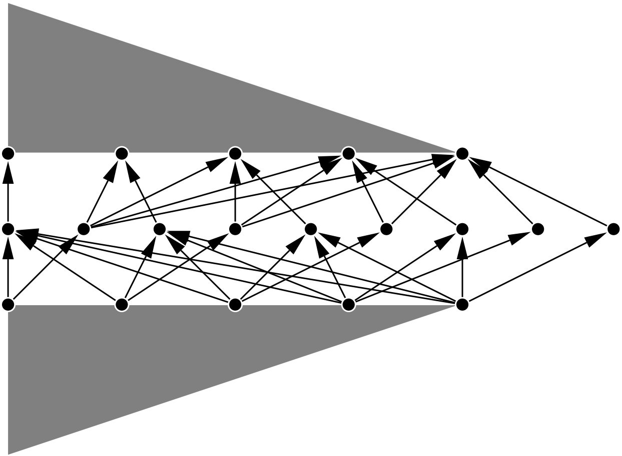


Combining the two types of minimisation

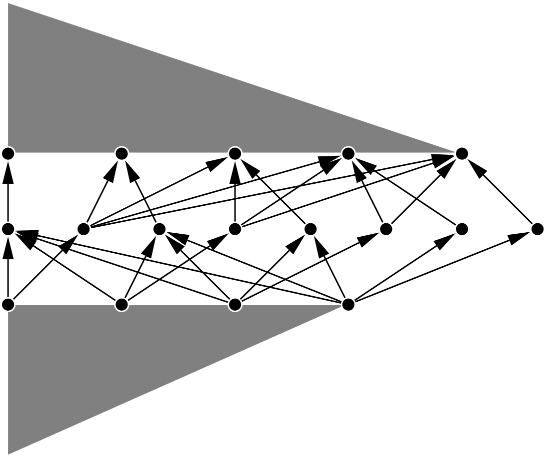
In a real-life application, one might wish to combine the two types of bisimulation to obtain even smaller output automata.



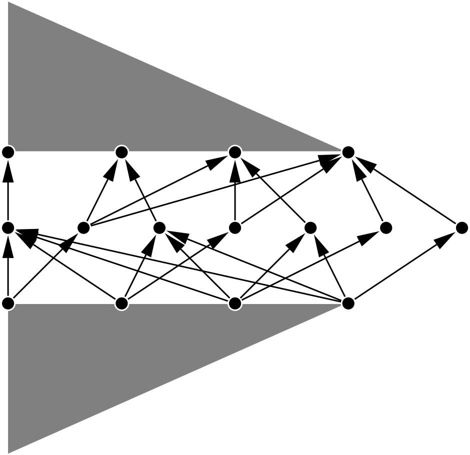
Combining the two types of minimisation



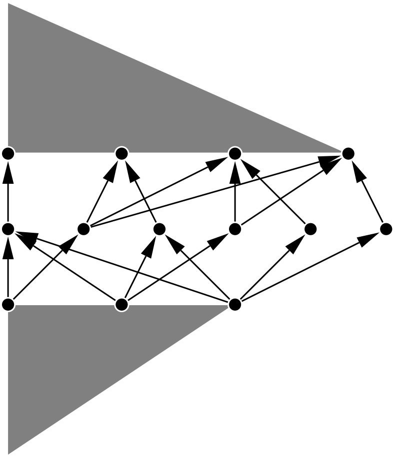
Combining the two types of minimisation



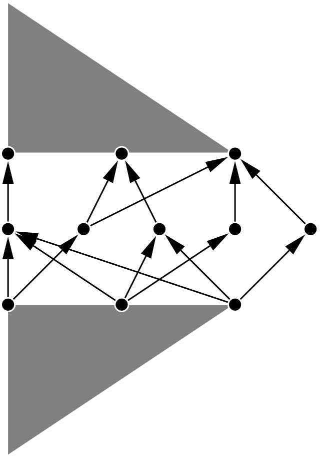
Combining the two types of minimisation



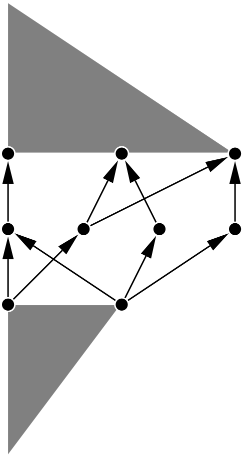
Combining the two types of minimisation



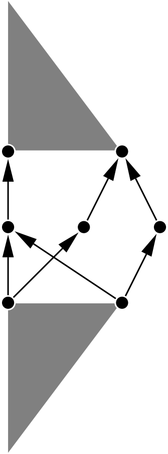
Combining the two types of minimisation



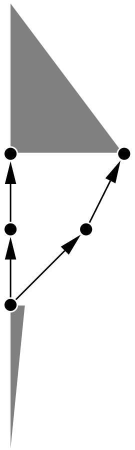
Combining the two types of minimisation



Combining the two types of minimisation



Combining the two types of minimisation



Combining the two types of minimisation

Partition refinement algorithms in general

The Coarsest Partition Problem

Given a transition system (Q, δ) and a condition c, find the coarsest partition of Q that meets with c.

1. Let the initial partition P_0 be $\{Q\}$.
2. Traverse the rules in δ, and
   - record the "behaviour" of each $q \in Q$ in the vector $v(q)_i$.
3. The partition P_{i+1} is obtained by bucket sorting each q in Q using $([q]_{P_i}, v(q)_i)$ as key.
4. if P_{i+1} and P_i coincide, then we are done, else, go to Step 2.

Time complexity

If δ is deterministic, then we can use the "process the smaller half" strategy by J. E. Hopcroft. In this case, we only have to consider a total of $\mathcal{O}(m \log n)$ rules, counting repetitions [Hopcroft, 1971].

If δ is nondeterministic, then we must also use a counting argument by Paige & Tarjan. [Paige and Tarjan, 1987].

Time complexity

Let r be the maximum rank of the input alphabet, let m be the number of transitions, and let n be the number of states.

- The forward algorithm runs in time $\mathcal{O}(r\, m \log n)$, and
- the backward algorithm runs in time $\mathcal{O}(r^2\, m \log n)$.

AKH bisimulation

An equivalence relation $\simeq$ is an AKH bisimulation if

- the relation respects the final states, and
- the fact that $p \simeq q$ and there is a rule

$$f(p_1, \ldots, p_{i-1}, p, p_i, \ldots, p_k) \to p_{k+1} \quad ,$$

where $i \in \{1, \ldots, k\}$, implies that there is also a rule

$$f(q_1, \ldots, q_{i-1}, q, q_i, \ldots, q_k) \to q_{k+1} \quad ,$$

s.t. $p_j \simeq q_j$, for every $j \in \{1, \ldots, k+1\} \setminus \{i\}$, and vice versa.

Comparison

Forward bisimulation . . .

- coincides with the standard minimisation algorithm when the input automaton is deterministic, and
- is a factor r easier to compute than both AKH bisimulation and backward bisimulation.

Backward bisimulation . . .

- is no harder to compute than AKH bisimulation, and
- produces, in the general case, smaller output automata than both forward and AKH bisimulation.

An NLP application

Problem

Compile a large set of syntactic trees into a language model.

An NLP application

Problem

Compile a large set of syntactic trees into a language model.

Instantiation

- Samples are taken from the Penn Treebank corpus of syntactically bracketed English news text.
- Fta as language model.

An NLP application

Problem

Compile a large set of syntactic trees into a language model.

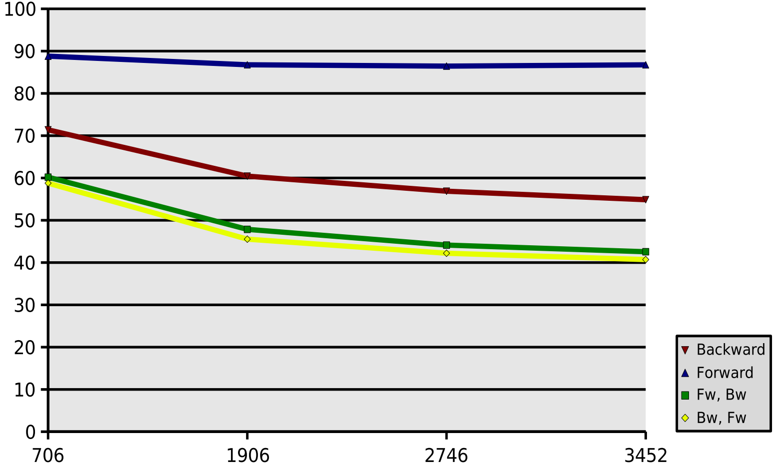
Instantiation

- Samples are taken from the Penn Treebank corpus of syntactically bracketed English news text.
- Fta as language model.

Solution

1. First, construct a trivial automaton from the sample set,
2. next, apply implementations of the minimisation algorithms.

Experimental results



Work in progress

Forward and backward bisimulation can also be defined for weighted tree automata.

- Leads to $\mathcal{O}(mnr)$ minimisation algorithms for general semirings, but
- $\mathcal{O}(r^2 m \log n)$, $\mathcal{O}(rm \log n)$ if the underlying algebraic structure is cancellative.

Future work includes

- weight pushing, and
- a more thorough study of the interaction between forward and backward bisimulation.