Advanced Automata Minimization

Lorenzo Clemente[1]   Richard Mayr[2]

[1]LaBRI, University of Bordeaux I, France
[2]LFCS, University of Edinburgh, UK

Highlights 2013

Computationally Hard Automata Problems

- Nondeterministic Büchi Automata
- NFA: Nondeterministic finite-word automata

PSPACE-complete problems.

Minimization: Given an automaton A. What is the minimal size of an automaton A' s.t. $\mathcal{L}(A) = \mathcal{L}(A')$?
(The minimal-size automaton for a given language is not unique, in general.)
In practice: Find a smaller A', not necessarily the smallest.

Inclusion: Given two automata A, B. Is $\mathcal{L}(A) \subseteq \mathcal{L}(B)$?
Related problems: Equivalence, Universality

PSPACE-completeness is no reason to despair.
Think of NP-complete problems and SAT-solvers.

Automata Minimization

- Given an automaton A. Find a smaller automaton A' s.t. $\mathcal{L}(A) = \mathcal{L}(A')$. (Not necessarily the smallest.)
- Algorithmic tradeoff between minimization effort and time for subsequent computations.
- Extensive minimization only worthwhile if hard questions are to be solved, e.g., inclusion, equivalence, universality, LTL model-checking.

Minimization Techniques

- Removing dead states. Remove states that cannot be reached, and states that cannot reach any accepting loop. (Trivial.)
- Quotienting. Find an equivalence relation $\equiv$ on the set of states. Merge equivalence classes into single states, inheriting transitions, and obtain a smaller automaton $A/\equiv$.
  If $\mathcal{L}(A/\equiv) = \mathcal{L}(A)$ then $\equiv$ is called good for quotienting (GFQ).
- Transition pruning. Some transitions can be removed without changing the language. This yields new dead states that can be removed.
  But how to find these superfluous transitions, without trial and error?
  Idea: Find a suitable relation R to compare transitions.
  Remove all transitions that are R-smaller than some other transition.
  If this preserves the language then R is called good for pruning (GFP).
  Problem: Relation R might be hard to compute. Removing transitions might change R. Need to remove transitions **in parallel**.

Minimization Techniques

- Removing dead states. Remove states that cannot be reached, and states that cannot reach any accepting loop. (Trivial.)
- Quotienting. Find an equivalence relation $\equiv$ on the set of states. Merge equivalence classes into single states, inheriting transitions, and obtain a smaller automaton $A/\equiv$.
  If $\mathcal{L}(A/\equiv) = \mathcal{L}(A)$ then $\equiv$ is called good for quotienting (GFQ).
- Transition pruning. Some transitions can be removed without changing the language. This yields new dead states that can be removed.
  But how to find these superfluous transitions, without trial and error?
  Idea: Find a suitable relation R to compare transitions.
  Remove all transitions that are R-smaller than some other transition.
  If this preserves the language then R is called good for pruning (GFP).
  Problem: Relation R might be hard to compute. Removing transitions might change R. Need to remove transitions **in parallel**.

Minimization Techniques

- Removing dead states. Remove states that cannot be reached, and states that cannot reach any accepting loop. (Trivial.)
- Quotienting. Find an equivalence relation $\equiv$ on the set of states. Merge equivalence classes into single states, inheriting transitions, and obtain a smaller automaton $A/\equiv$.
  If $\mathcal{L}(A/\equiv) = \mathcal{L}(A)$ then $\equiv$ is called good for quotienting (GFQ).
- Transition pruning. Some transitions can be removed without changing the language. This yields new dead states that can be removed.
  But how to find these superfluous transitions, without trial and error?
  Idea: Find a suitable relation R to compare transitions.
  Remove all transitions that are R-smaller than some other transition.
  If this preserves the language then R is called good for pruning (GFP).
  Problem: Relation R might be hard to compute. Removing transitions might change R. Need to remove transitions **in parallel**.

Minimization Techniques

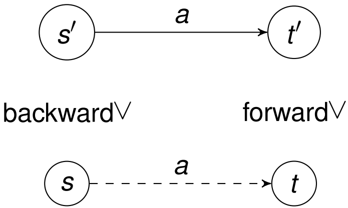
- Removing dead states. Remove states that cannot be reached, and states that cannot reach any accepting loop. (Trivial.)
- Quotienting. Find an equivalence relation $\equiv$ on the set of states. Merge equivalence classes into single states, inheriting transitions, and obtain a smaller automaton $A/\equiv$.
  If $\mathcal{L}(A/\equiv) = \mathcal{L}(A)$ then $\equiv$ is called good for quotienting (GFQ).
- Transition pruning. Some transitions can be removed without changing the language. This yields new dead states that can be removed.
  But how to find these superfluous transitions, without trial and error?
  Idea: Find a suitable relation R to compare transitions.
  Remove all transitions that are R-smaller than some other transition.
  If this preserves the language then R is called good for pruning (GFP).
  Problem: Relation R might be hard to compute. Removing transitions might change R. Need to remove transitions **in parallel**.

Minimization Techniques

- Removing dead states. Remove states that cannot be reached, and states that cannot reach any accepting loop. (Trivial.)
- Quotienting. Find an equivalence relation $\equiv$ on the set of states. Merge equivalence classes into single states, inheriting transitions, and obtain a smaller automaton $A/\equiv$.
  If $\mathcal{L}(A/\equiv) = \mathcal{L}(A)$ then $\equiv$ is called good for quotienting (GFQ).
- Transition pruning. Some transitions can be removed without changing the language. This yields new dead states that can be removed.
  But how to find these superfluous transitions, without trial and error?
  Idea: Find a suitable relation R to compare transitions.
  Remove all transitions that are R-smaller than some other transition.
  If this preserves the language then R is called good for pruning (GFP).
  Problem: Relation R might be hard to compute. Removing transitions might change R. Need to remove transitions **in parallel**.

Transition Pruning with Semantic Preorders

Compare transitions $s \xrightarrow{a} t$ and $s' \xrightarrow{a} t'$ by comparing their source and target.



If s' is backward-bigger than s, and
t' is forward-bigger than t then
consider $s' \xrightarrow{a} t'$ as bigger than $s \xrightarrow{a} t$ and
remove the superfluous transition $s \xrightarrow{a} t$.

But does this preserve the language?
Which semantic relations are suitable for backward-bigger and forward-bigger?

Comparing States of Automata

Simulation: $s \sqsubseteq t$ iff t can match the computation of s stepwise.
Simulation game: Spoiler moves $s \xrightarrow{a} s'$.
Duplicator replies $t \xrightarrow{a} t'$.
Next round of the game starts from s', t'.
Simulation preorder is polynomial.

Trace inclusion: $s \subseteq t$ iff t has at least the same traces as s.
Trace game: Spoiler chooses a trace $s \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \dots$.
Duplicator replies with a trace $t \xrightarrow{a_1} t_1 \xrightarrow{a_2} t_2 \dots$.
Trace inclusion is PSPACE-complete.

Trace inclusion is generally much larger than simulation, but hard to compute.

Backward simulation/traces defined similarly with backward steps.

Acceptance Conditions

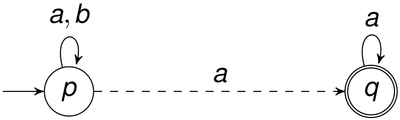
Direct: If Spoiler accepts then Duplicator must accept immediately.

Delayed: If Spoiler accepts then Duplicator must accept eventually (i.e., within finitely many steps in the future, but there is no fixed bound).

Fair: If Spoiler accepts infinitely often then Duplicator must accept infinitely often.
(This is a weaker condition than delayed. If Spoiler accepts only finitely often then Duplicator has no obligations.)

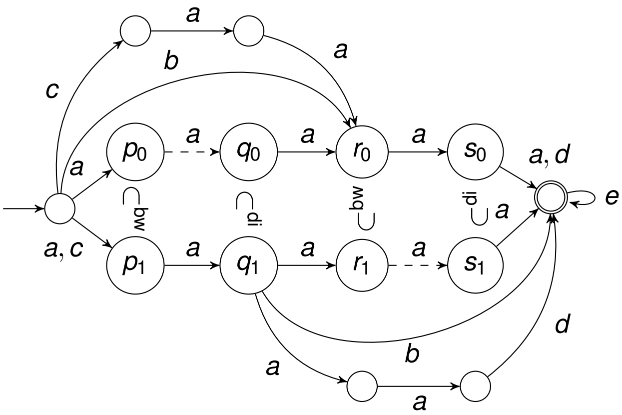
This yields semantic preorders of direct/delayed/fair simulation and trace inclusion.
Preorders induce equivalences by considering both directions.

Delayed/Fair Simulation is **not** Good-for-Pruning



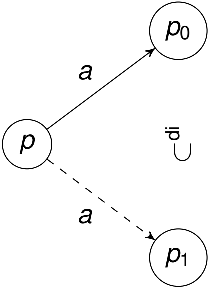
$q \sqsubset^{\mathrm{de}} p$, so the transition $p \xrightarrow{a} p$ looks larger than $p \xrightarrow{a} q$.
However, removing the dashed transition $p \xrightarrow{a} q$ makes the language empty.

Pruning with Direct Forward **and** Backward Trace Inclusion is Incorrect



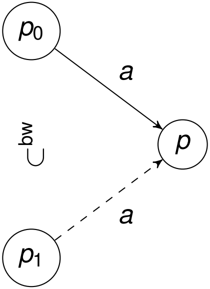
If the ‘smaller’ dashed transitions are removed then the word $aaaae^{\omega}$ is no longer accepted.

Pruning w.r.t. Direct Forward Trace Inclusion



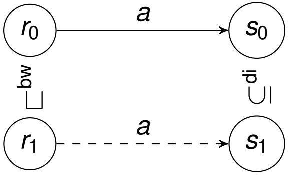
This generalizes [Bustan, Grumberg] who use direct forward simulation.

Pruning w.r.t. Direct Backward Trace Inclusion

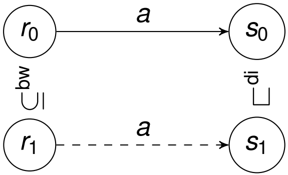


This generalizes [Bustan, Grumberg] who use direct backward simulation.

Pruning w.r.t. Direct Backward Simulation and Forward Trace Inclusion



Pruning w.r.t. Direct Backward Trace Inclusion and Forward Simulation



One can have backward simulation and forward trace-inclusion, or vice-versa, but **not both trace-inclusions**.

Quotienting

- Forward/backward direct simulation/trace-equivalence is good for quotienting (GFQ).
- Fair simulation/trace-equivalence is not GFQ.
- Delayed simulation is GFQ, but delayed trace inclusion is not GFQ.
- Delayed multipebble simulation [Etessami, Wilke, Schuller] allows Duplicator to hedge his bets in the simulation game, yielding a larger relation. GFQ, but hard to compute (exponential in the number of pebbles).

Computing Semantic Preorders

One would like to use

- Direct backward/forward trace inclusion for pruning (and quotienting).
- Multipebble delayed simulation for quotienting.

But these are hard to compute (PSPACE-complete membership problem).

Idea: Compute good under-approximations of these relations.

k-Lookahead-simulations:

- Play a simulation game where Duplicator has information about Spoiler's next k moves.
- Higher lookahead k yields larger relations, but is harder to compute.
- Many possible ways of defining lookahead. Most are very bad.
- Idea: Degree of lookahead is dynamically under the control of Duplicator, i.e., use only as much as needed (up-to k).
  Efficient computation **and** large relations.

Benchmarks

GOAL: Best effort of previous methods. Quotienting/pruning w.r.t. backward/forward simulation. Delayed simulation quotienting. Fair simulation minimization of [Gurumurthy, Bloem, Somenzi].

Heavy-12: Our transition pruning and quotienting methods with lookahead simulations of lookahead 12. Much faster than GOAL.

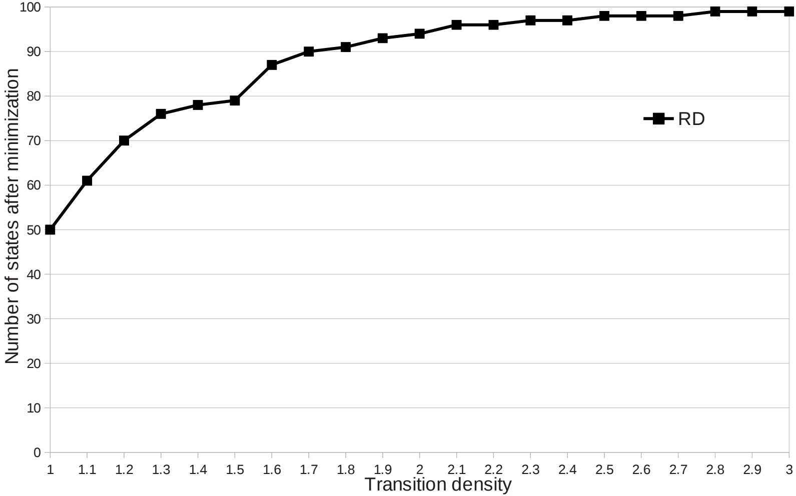
Test cases.

Protocols: Automata derived from protocols like Peterson, Fischer, Phils, Bakery, Mcs. Heavy-12 minimizes better and faster than GOAL; see table in paper.

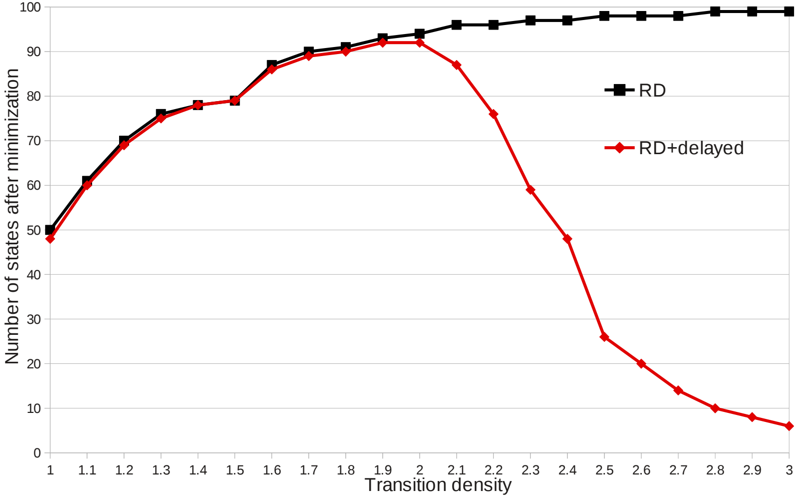
LTL formulae: Consider large (size 70) LTL-formulae, transformed into Büchi automata by LTL2BA and minimized by GOAL.
82% can be minimized further by Heavy-12.
Average reduction ratio 0.76 for states and 0.68 for transitions.

Tabakov-Vardi random automata: Binary alphabet.
Transition density = #transitions/(# states * # symbols).
Acceptance density 0.5 (does not matter much).

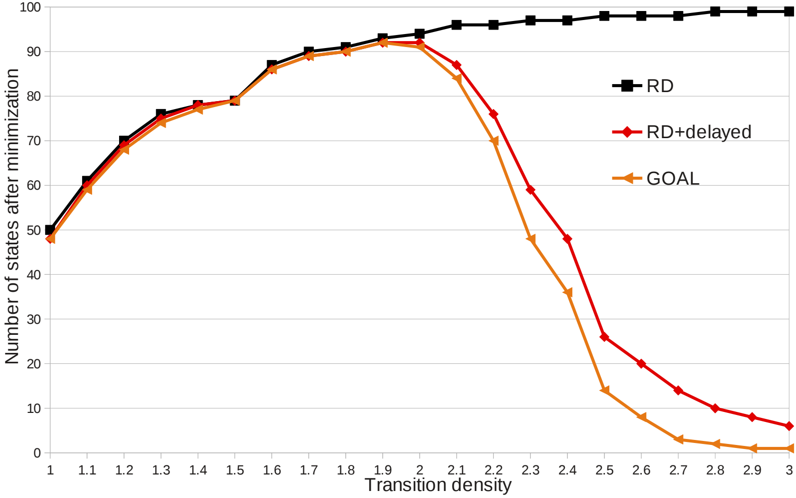
Benchmark: Remove dead states



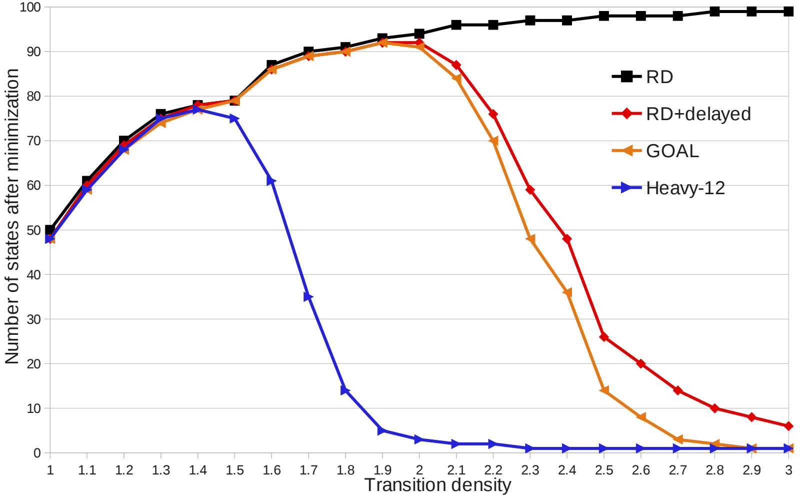
Benchmark: Remove dead + quotient with delayed sim



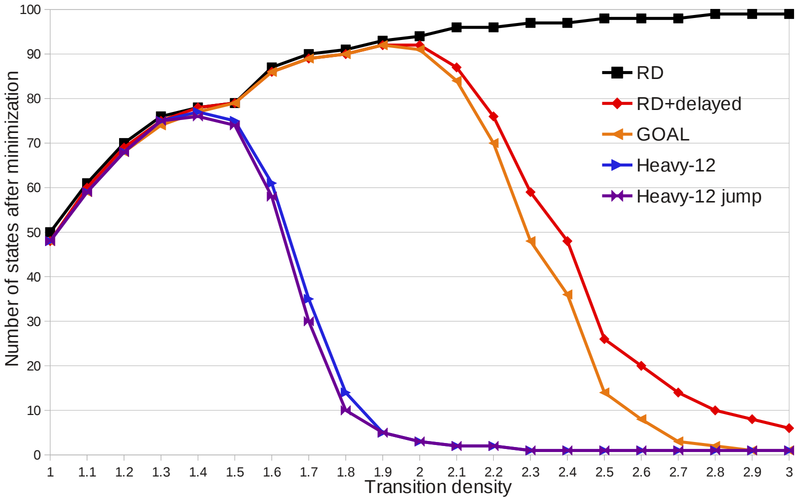
Benchmark: Best effort of GOAL



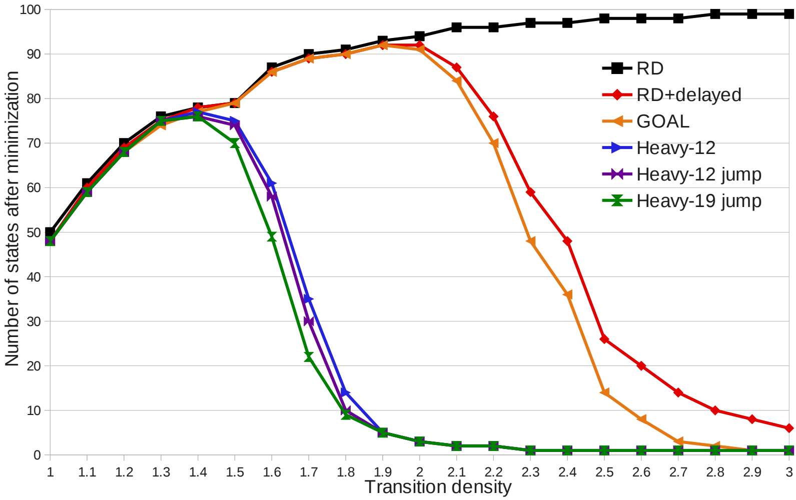
Benchmark: Heavy-12



Benchmark: Heavy-12 plus jumping simulation



Benchmark: Best. Lookahead 19 plus jumping simulation



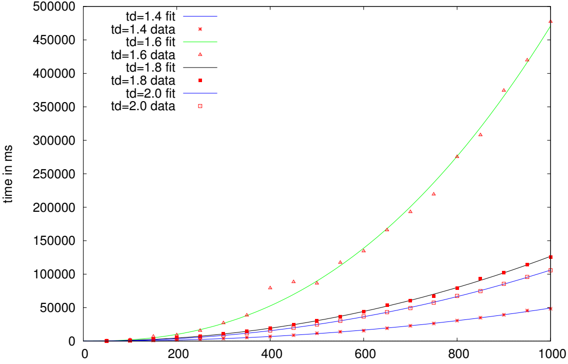
Scalability: Almost quadratic

Minimize Tabakov-Vardi automata with transition density 1.4, 1.6, 1.8, 2.0.

The size increases from $n = 50$ to $n = 1000$ states.

Plot the time and compute the best fit of the function $time = a * n^b$.

This yields exponents b between 2.05 and 2.39. Almost quadratic average-case complexity.



Average time for Heavy-12 minimization for td=1.4,1.6,1.8, 2.0 with y=a*x^b fit.

Language Inclusion Checking

Checking language inclusion $\mathcal{L}(A) \subseteq \mathcal{L}(B)$ of Büchi automata.

- Minimize A and B together.
- (Generalized) simulations can witness inclusion already at this stage (if inclusion holds). This happens very often.
- Additional pruning techniques and witnessing inclusion by jumping lookahead fair simulation.
- If inclusion was not proven yet, then use a complete technique on the now smaller instance A', B'.

Can check inclusion of Tabakov-Vardi Büchi automata with 1000 states. Success rate $98\% - 100\%$, depending on density. Much better than previous techniques.

Conclusion

- Minimize automata with transition pruning, not only quotienting.
- Compute good approximations of trace-inclusion and multipebble-simulation by lookahead-simulations.
- Much better automata minimization.
- Can check inclusion for much larger Büchi automata.
- Techniques carry over to NFA, but
  - Good NFA minimization.
  - NFA inclusion/equivalence checking: Since NFA are simpler, computing global relations like simulation is not always worth the effort.
    $\longrightarrow$ Talk by Bonchi and Pous.
- Links and tools available at **www.languageinclusion.org**