Techniques to solve computationally hard problems in automata theory

Richard Mayr

University of Edinburgh, UK

IST Vienna, 4. Nov. 2014

Resources: www.languageinclusion.org

Outline

1. Computationally Hard Automata Problems
2. Antichain Techniques
3. Bisimulation Modulo Congruence
4. Automata Minimization
5. Benchmarks
6. Language Inclusion Checking by Minimization

Automata

We consider automata which are

- Nondeterministic
- Finite-state
- Accepting words (for generalization to trees see libvata, etc.)

Finite words vs. infinite words

- NFA: Automata accepting finite words. Like in undergraduate class. Regular languages.
- Büchi automata: Automata accepting infinite words.
  Word $w \in \Sigma^\omega$ is accepted iff
  there is a run on w that visits an accepting state infinitely often.
  ($\exists$ run ρ on w s.t. $inf(\rho) \cap F \neq \emptyset$.)
  ω-regular languages.
  Büchi automata are not determinizable, but still closed under complement.

Hard 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.)

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

Equivalence: Given two automata A, B. Is $\mathcal{L}(A) = \mathcal{L}(B)$?

Universality: Given an automaton A. Is $\mathcal{L}(A) = \Sigma^\omega$ (resp. Σ^*) ?

All these problems are PSPACE-complete.

But this is no reason not to solve them.
Think of NP-complete problems and SAT-solvers.

Automata and Software Verification

- Comparing control-flow graphs of specification and implementation.
  Does the implementation only behave in a way allowed by the specification?
  $\longrightarrow$ Language inclusion problem.
- Termination analysis.
  Size-change termination proofs work by abstracting data via a function. Does this function decrease infinitely often along **every** infinite computation? Then the program must terminate.
  $\longrightarrow$ Language inclusion problem.
- Model checking.
  Temporal logic formulae translated into automata (e.g., LTL).
  Or representing large sets of configurations by regular languages.
  $\longrightarrow$ Automata minimization problem.
- Decision procedures for logical theories.
  Automata represent denotations of formulae. Small is beautiful.
  $\longrightarrow$ Automata minimization problem.

Antichain Techniques

Universality problem for NFA $A = (Q, \Sigma, \delta, q_0, F)$. Is $\mathcal{L}(A) = \Sigma^*$?
Search for a counterexample, i.e., a word that is not accepted.
Powerset construction on the fly. Start from $\{q_0\}$ and explore reachable macrostates $S \subseteq Q$. If $S \cap F = \emptyset$ then S is a rejecting macrostate, and we have found a counterexample.
The number of macrostates is exponential. How to narrow the search space?
Subsumption: A special case of logical redundancy.
Suppose we have two macrostates S, S' with

$$S \subset S'$$

Then every counterexample (i.e., reachable rejecting macrostate) that can be found from S' can also be found from S.
Why? The successor relation on macrostates is monotone w.r.t. set inclusion.
So S is “better” than S', i.e., S subsumes S' and S' can be discarded from the search.

Antichain Techniques

Antichain Algorithm

Search reachable macrostates and keep a record of the states explored so far.
Discard all macrostates that are subsumed by previously generated ones.
If you find a macrostate state S with $S \cap F = \emptyset$ return false.
Otherwise, return true.

Since subsumed macrostates are discarded, all recorded macrostates are incomparable, i.e., they form an antichain w.r.t. the given relation that is used to compare them.

The hope is that, for the given automaton, the antichain is small.

Better subsumption relations

How much subsumption helps depends on how large the subsumption relation is, i.e., how many macrostates are comparable.
Larger subsumption relation $\longrightarrow$ Smaller antichain.
Can one use more than just set inclusion?
Suppose we have a relation $\sqsubseteq$ on Q (i.e., on states, not macrostates) s.t. $q \sqsubseteq q' \;\Rightarrow\; \mathcal{L}(q) \subseteq \mathcal{L}(q')$.
Lift this relation to macrostates (à la Plotkin):

$$S \sqsubseteq_{\forall\exists} S' \;\Leftrightarrow\; \forall q \in S. \exists q' \in S. \, q \sqsubseteq q'$$

Since $\mathcal{L}(S) = \bigcup_{q \in S} \mathcal{L}(q)$ we have that

$$S \sqsubseteq_{\forall\exists} S' \;\Rightarrow\; \mathcal{L}(S) \subseteq \mathcal{L}(S')$$

For finding counterexamples to universality, S subsumes S', because on macrostates (i.e., DFA) language inclusion is monotone w.r.t. transition steps.

Approximating language inclusion

Ideally, we want to find a relation $\sqsubseteq$ on Q s.t.

$$q \sqsubseteq q' \;\Rightarrow\; \mathcal{L}(q) \subseteq \mathcal{L}(q')$$

It should be

- As large as possible.
- Efficiently computable.

These are conflicting goals.

- Smallest relation: Just identity. Very efficient, but then $\sqsubseteq_{\forall\exists}$ is just set inclusion. (I.e., we get basic subset-subsumption as before).
- Largest relation: Language inclusion itself. PSPACE-complete. (We are running around in circles, since language inclusion is the problem we want to solve.)

Compromise: Simulation preorder. q' needs to imitate the behavior of q stepwise. PTIME-computable, but larger than identity.
Generalized simulations (multipebble, lookahead) trade higher computation time for a larger relation. (Later in this talk.)

Antichain Techniques for Büchi Automata

Checking universality of a nondeterministic Büchi automaton A. By a theorem of Büchi, we have

$$\mathcal{L}(A) \neq \Sigma^\omega$$

iff

$$\exists w_1, w_2 \in \Sigma^+ .\ w_1 (w_2)^\omega \notin \mathcal{L}(A)$$

So we can limit the search to a regular counterexample to universality.

Ramsey-based technique: Generate graphs $G \subseteq Q \times Q$ that characterize the behavior of A.

Intuition: For $L \subseteq \Sigma^+$, G_L contains an edge (q, q') iff $\exists w \in L.\, q \xrightarrow{w} q'$.

A counterexample is witnessed by two graphs G_{L_1} and G_{L_2} that satisfy certain conditions.

Explore the space of these graphs and use a subsumption relation to narrow the search space.

Subsumption relations based on backward/forward simulation by [Mayr, Abdulla, Chen, Clemente, Holik, Hong, Vojnar: CONCUR'11]. Very technical.

Antichain summary

- A glorified search for a counterexample.
- Use subsumption relation to compare elements and prune the search space.
- Comparison is one-on-one. Discard one element, because one single other element is better.
- Stored/explored elements from an antichain w.r.t. the subsumption relation.
- Bigger subsumption relation makes more elements comparable. Fewer elements to compare. Shorter antichain on given instance.

Previous slides explained the concept for universality testing, but it generalizes easily to language inclusion testing $\mathcal{L}(A) \subseteq \mathcal{L}(B)$. Explored elements additionally contain states of A.

Bisimulation Modulo Congruence [Bonchi-Pous:POPL'13]

Given an NFA A and states $q_1, q_2 \in Q$. Check $\mathcal{L}(q_1) = \mathcal{L}(q_2)$.

Explore pairs of macrostates (S_1, S_2) reachable from $(\{q_1\}, \{q_2\})$. They need to satisfy $\mathcal{L}(S_1) = \mathcal{L}(S_2)$ or else there is a counterexample. In particular, S_1, S_2 need to agree on acceptance.

Maintain sets of macrostates *Explored* and *toExplore*.

Main idea to reduce the search space: The set of pairs *Explored*, *toExplore* induces a congruence $\equiv$. If for a given pair of macrostates (S_1, S_2) we have $S_1 \equiv S_2$, then it can be discarded. Why? Either $\mathcal{L}(S_1) = \mathcal{L}(S_2)$ or a shorter counterexample can be found elsewhere.

Example: Let $(X_1, X_2), (Y_1, Y_2) \in$ *Explored*. Then $X_1 \cup Y_1 \equiv X_2 \cup Y_2$.

How to check the relation $\equiv$? Consider *Explored*, *toExplore* as a set of rewrite rules and reduce pairs of macrostates to a normal form.

Antichains vs. Bisimulation Modulo Congruence

Both are a glorified search for a counterexample.

Antichains	**Congruence**
One element subsumed by one other	One element subsumed by combination of many others
Subsumption easy to check	Subsumption computationally harder
Fewer elements discarded	More elements discarded
Hope for short antichain	Hope for small congruence base
NFA and Büchi automata	Only NFA (so far)

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

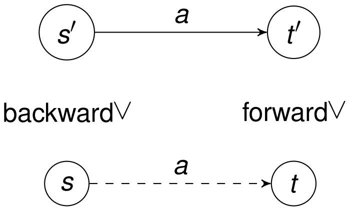
- Removing dead states. Remove states that cannot be reached, and states that cannot reach any accepting loop. (Trivial.)
- Quotienting. Find an equivalence relation $\equiv$ on the set of states. Merge equivalence classes into single states, inheriting transitions, and obtain a smaller automaton $A/\equiv$.
  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 $s \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \ldots$.
Duplicator replies with a trace $t \xrightarrow{a_1} t_1 \xrightarrow{a_2} t_2 \ldots$.
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

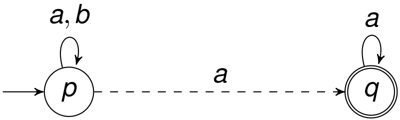
Direct: If Spoiler accepts then Duplicator must accept immediately.

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.

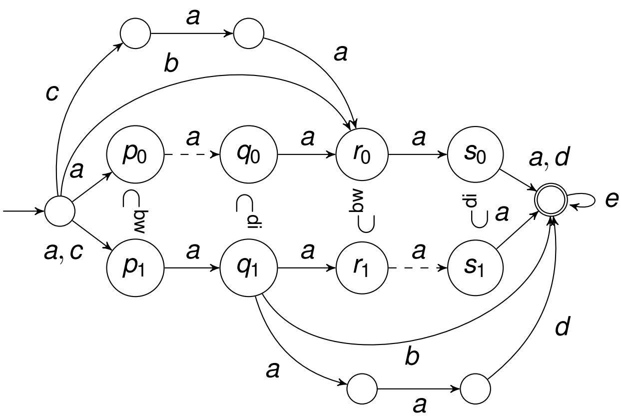
Delayed/Fair Simulation is **not** Good-for-Pruning



$q \sqsubset^{\mathrm{de}} p$, 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.

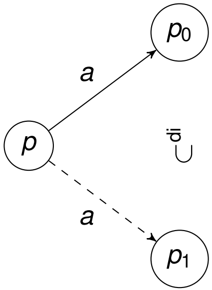
Special case: Suppose the larger remaining transition is **transient** (can be used at most once). Then delayed/fair simulation (and even language inclusion) is good for pruning.
Let $x \xrightarrow{a} p$ and $x \xrightarrow{a} q$ s.t. $p \subset^{\mathrm{f}} q$ and $x \xrightarrow{a} q$ is transient, then $x \xrightarrow{a} p$ can be removed.

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



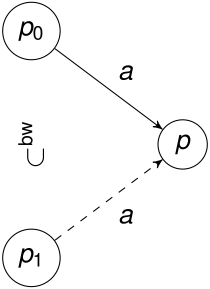
If the 'smaller' dashed transitions are removed then the word $aaaaae^{\omega}$ is no longer accepted.

Pruning w.r.t. Direct Forward Trace Inclusion



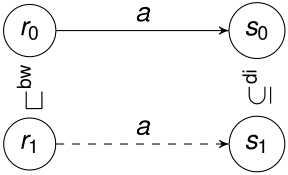
This generalizes [Bustan, Grumberg] who use direct forward simulation.

Pruning w.r.t. Direct Backward Trace Inclusion

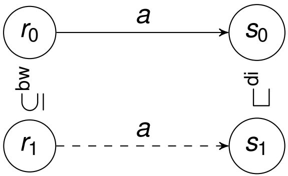


This generalizes [Bustan, Grumberg] who use direct backward simulation.

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**.
Simulation is preserved stepwise, unlike trace-inclusion.

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] 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.

Generalized Simulations

Simulations can be seen as polynomial-size locally checkable certificates, witnessing the larger relation of trace-inclusion.

Polynomial time computable, but normally much smaller than trace-inclusion.

Extensions:

Multipebble simulation: [Etessami]. Duplicator has several pebbles and can hedge his bets, i.e., keep his options open.
Exponential time (and space!) in the number of pebbles used.
Even for just 2 pebbles, one needs at least cubic time and space.
Not practical for large automata.

Lookahead Simulations

k-step simulation: Spoiler announces k steps. Duplicator replies with k steps.
Space efficient computation. Too many cases of k steps. Too inflexible: Lookahead is not used where it is most needed.

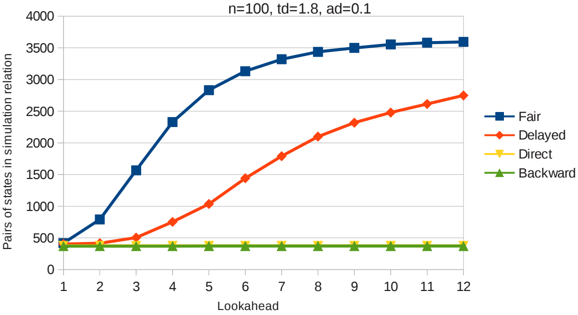
k-continuous simulation: Duplicator is always kept informed of Spoiler's next k steps.
Larger relation.
Still too inflexible: lookahead often used where it is not needed.
Hard to compute: Game graph size $n^2 * d^k$. Too much space/time.

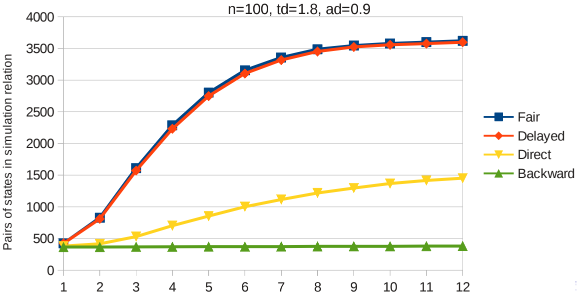
k-lookahead simulation: Spoiler announces k steps. Duplicator chooses $m : 1 \leq m \leq k$ and replies to the first m steps.
Remaining Spoiler steps are forgotten. Next round.
Space efficient. Lookahead dynamically under Duplicator's control and used where it is most needed.
Computational advantage: Spoiler builds his long move incrementally. Duplicator can reply to a prefix and win the round immediately. The maximal lookahead is rarely used.

Lookahead vs. Pebbles

- Neither k-lookahead simulation nor k-pebble simulation is transitive in general.
  Use their transitive closure.
- k-lookahead simulation and k-pebble simulation are incomparable, but n-pebble simulation subsumes all others (for automata with n states).
  Even stronger incomparable cases: 2 pebbles and arbitrary lookahead.
  $n-1$ pebbles and 2-lookahead.
- One can express k-lookahead simulation in terms of pebbles.
  Duplicator can use the maximal number n of pebbles, but
  after at most k steps (earlier is allowed) Duplicator has to commit to just one pebble. Then he can use maximal pebbles again.
- Combinations are possible, e.g., 2-pebble k-lookahead simulation, but on average it is not worth it. On average, a limited time budget is better invested in a higher lookahead, not pebbles.

Density of Simulation 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.

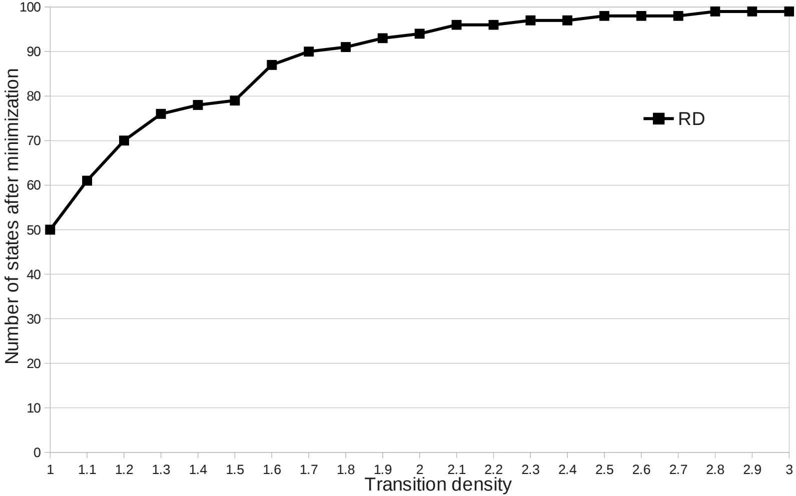
Test cases.

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

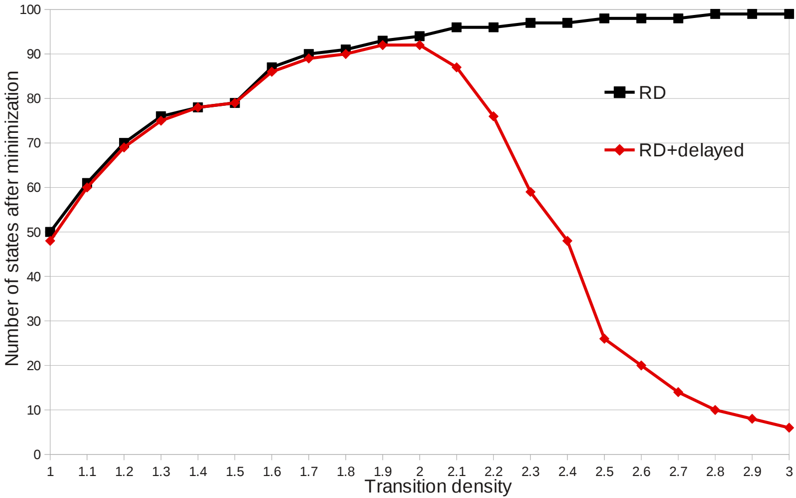
LTL formulae: Consider large (size 70) LTL-formulae, transformed into Büchi automata by LTL2BA and minimized by GOAL.
82% can be minimized further by Heavy-12.
Average reduction ratio 0.76 for states and 0.68 for transitions.

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

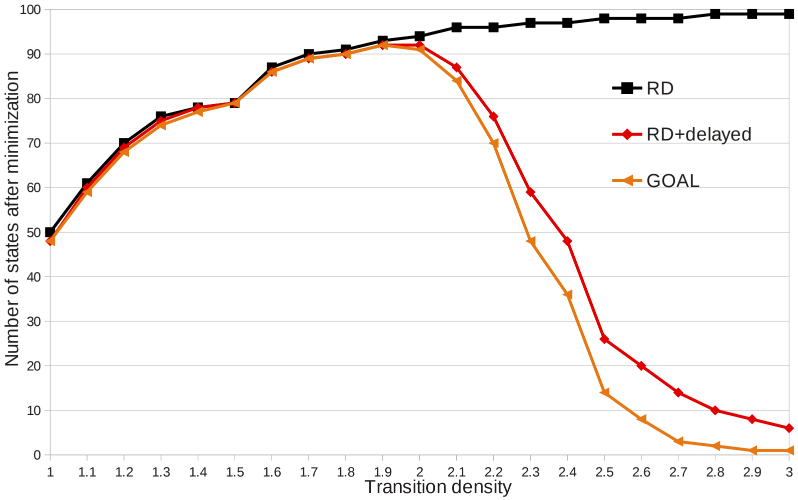
Benchmark: Remove dead states



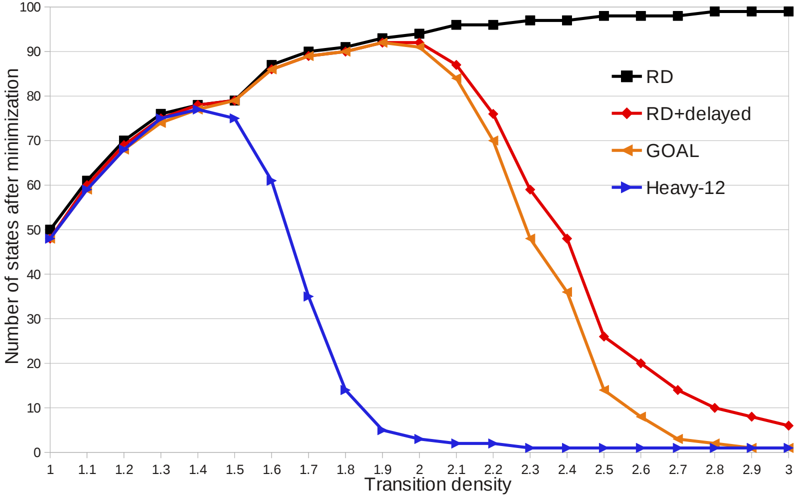
Benchmark: Remove dead + quotient with delayed sim



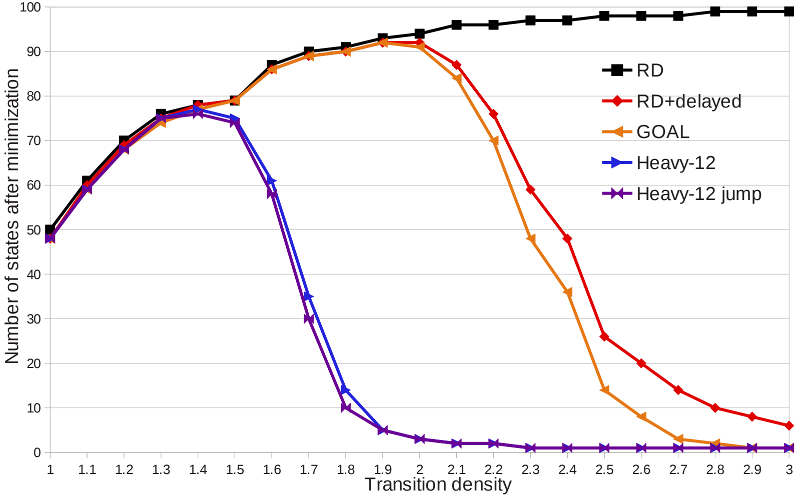
Benchmark: Best effort of GOAL



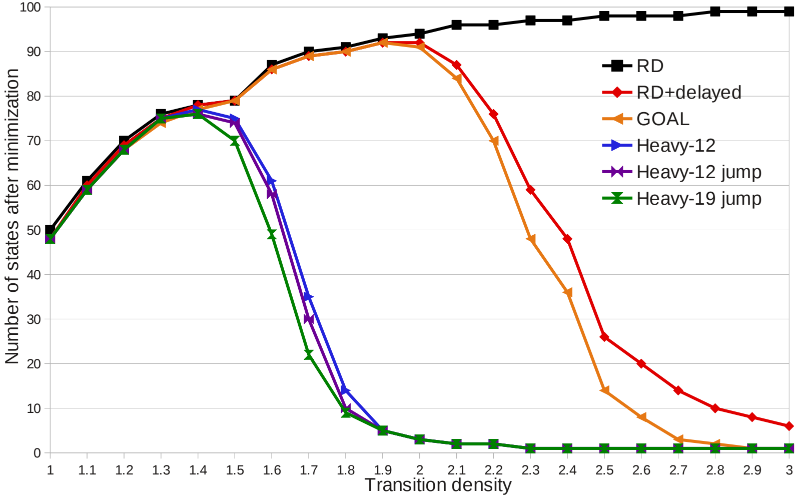
Benchmark: Heavy-12



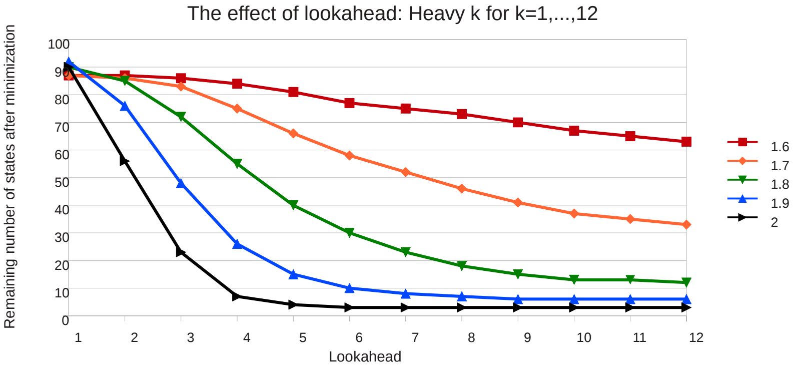
Benchmark: Heavy-12 plus jumping simulation



Benchmark: Best. Lookahead 19 plus jumping simulation

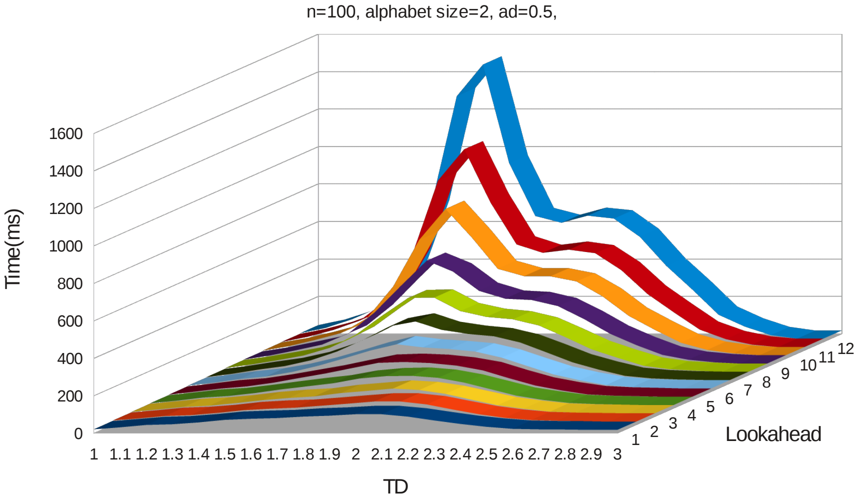


The Effect of Lookahead

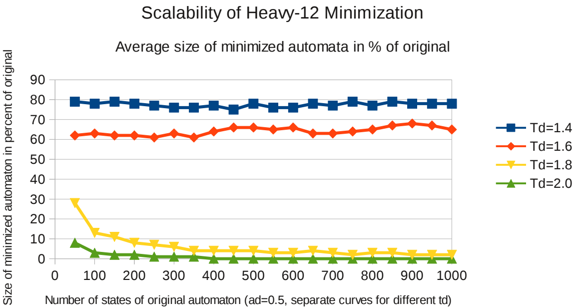


Computation Time

Average computation time for minimization with Heavy-k



Scalability



Minimization of Tabakov-Vardi random automata with $ad = 0.5$, $|\Sigma| = 2$, and increasing $n = 50, 100, \ldots, 1000$. Different curves for different td. Average size of the Heavy-12 minimized automata, in percent of their original size.

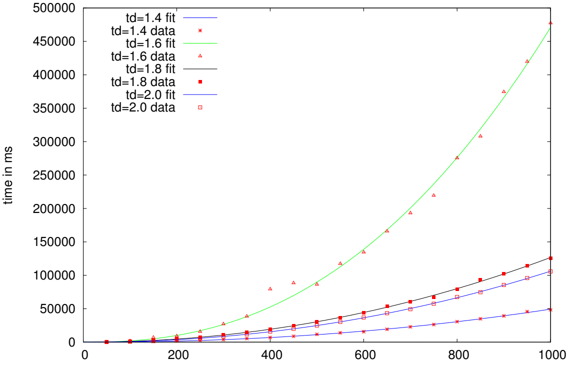
Scalability (cont.)

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.

- Explicit or implicit complementation of B.
- Rank-based, Ramsey-based, Slice-based, Safra-Piterman.
- A glorified search for a counterexample to inclusion.
- On-the fly constructions.
- Subsumption techniques. We don't need to explore X, because we explore Y which is better.
  Forward/backward simulation preorders used to help in subsumption.

Language Inclusion Checking by Minimization

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: Discard some parts of A and B that don't affect a counterexample (even if this changes the languages of A, B).
- Witnessing inclusion by jumping lookahead fair simulation. Duplicator can jump to states that are (direct/counting/segmented) backward-trace larger than his current state.
- 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.

NFA

- Minimization/Inclusion techniques carry over to NFA, using only direct simulation.
- Transform automata into a form with only one absorbing accepting state. Otherwise the direct (lookahead-)simulations are too small.
- Simulations are global relations: Since NFA's are conceptually simpler, there is more competition from local techniques.
- Antichain techniques: Search for a counterexample with subsumption (e.g., libvata on word/tree automata). Worst-case exponential time/space. Time is quadratic in the final size of the store.
- Bisimulation modulo congruence (Bonchi & Pous). Current store can collectively subsume a new element. Worst-case exponential time/space. Cubic time in the final size of the store, but much smaller store size.

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.
- Links and tools available at **www.languageinclusion.org**
  Büchi automata, NFA, Tree-automata.