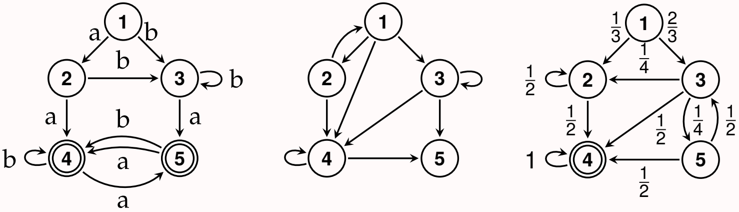
Fast Coalgebraic Bisimilarity Minimization

Jules Jacobs
Radboud University

Thorsten Wißmann
Radboud University
$\rightarrow$
Friedrich-Alexander-Universität
Erlangen-Nürnberg

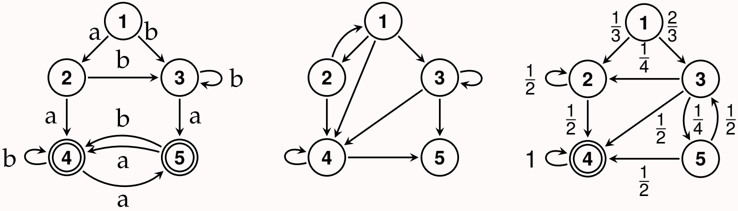
The Automaton Zoo

Deterministic finite automata, tree automata, (labeled) transition systems, weighted and probabilistic automata, Markov decision processes, ...



The Automaton Zoo

Deterministic finite automata, tree automata, (labeled) transition systems, weighted and probabilistic automata, Markov decision processes, ...

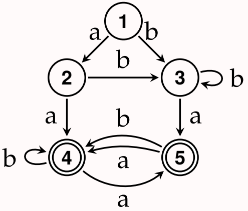


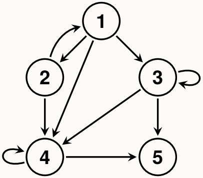
Automaton Minimization

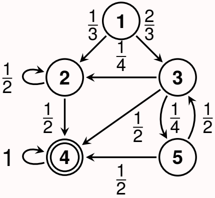
Find and merge behaviorally equivalent states

The Automaton Zoo

Deterministic finite automata, tree automata, (labeled) transition systems, weighted and probabilistic automata, Markov decision processes, ...







Automaton Minimization

Find and merge behaviorally equivalent states

Coalgebraic Bisimilarity Minimization

Algorithms that work for a general class of F-automata

Our contribution

a **fast** and **general** algorithm for minimizing automata

Our contribution

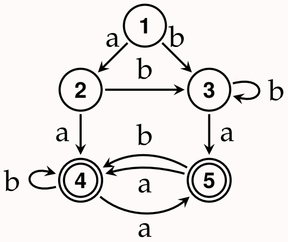
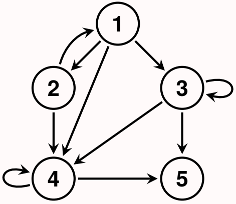
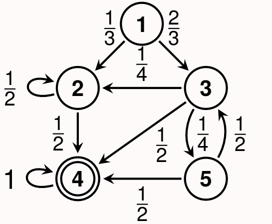
a **fast** and **general** algorithm for minimizing automata

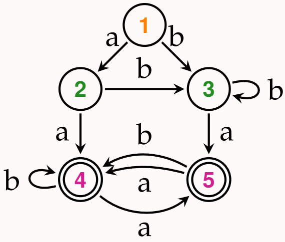
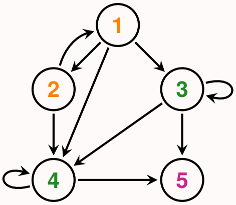
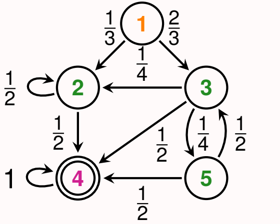
- *General:* works for any computable coalgebra
- *Decent asymptotic complexity:* $O(\phi_F \cdot m \log n)$
- *Fast in practice:* no penalty for generality
- *Low memory usage:* important for large automata

Examples of Coalgebraic Automata

Automaton type	**Equivalence**	**Functor** $F(X)$
DFA	Language Equivalence	$2 \times A^X$
Transition Systems	Strong Bisimilarity	$\mathcal{P}(X)$
LTS	Strong Bisimilarity	$\mathcal{P}(A \times X)$
Weighted Systems	Weighted Bisimilarity	$M^{(X)}$
Markov Chain	Probabilistic Bisimilarity	$A \times \mathcal{D}(X)$
MDP	Probabilistic Bisimilarity	$\mathcal{P}(\mathcal{D}(X))$
Weighted Tree Automata	Backwards Bisimilarity	$M^{(\Sigma X)}$
Monotone Neigh. Frames	Monotone Bisimilarity	$\mathcal{N}(X)$
$\vdots$	$\vdots$	$\vdots$

Automaton types compose: $F \circ G, F + G, F \times G, \ldots$

<table>
<tr><th>DFA</th><th>Transition system</th><th>Markov chain</th></tr>
<tr>
<td>
</td>
<td>
</td>
<td>
</td>
</tr>
<tr><td>$F(X) = \{\mathsf{F}, \mathsf{T}\} \times X \times X$</td><td>$F(X) = \mathcal{P}_{\mathsf{f}}(X)$</td><td>$F(X) = \{\mathsf{F}, \mathsf{T}\} \times \mathcal{D}(X)$</td></tr>
<tr>
<td>$\mathbf{1} \mapsto (\mathsf{F}, \mathbf{2}, \mathbf{3})$
$\mathbf{2} \mapsto (\mathsf{F}, \mathbf{4}, \mathbf{3})$
$\mathbf{3} \mapsto (\mathsf{F}, \mathbf{5}, \mathbf{3})$
$\mathbf{4} \mapsto (\mathsf{T}, \mathbf{5}, \mathbf{4})$
$\mathbf{5} \mapsto (\mathsf{T}, \mathbf{4}, \mathbf{4})$</td>
<td>$\mathbf{1} \mapsto \{\mathbf{2}, \mathbf{3}, \mathbf{4}\}$
$\mathbf{2} \mapsto \{\mathbf{1}, \mathbf{4}\}$
$\mathbf{3} \mapsto \{\mathbf{3}, \mathbf{4}, \mathbf{5}\}$
$\mathbf{4} \mapsto \{\mathbf{4}, \mathbf{5}\}$
$\mathbf{5} \mapsto \{\ \}$</td>
<td>$\mathbf{1} \mapsto (\mathsf{F}, \{\mathbf{2}\colon \frac{1}{3}, \mathbf{3}\colon \frac{2}{3}\})$
$\mathbf{2} \mapsto (\mathsf{F}, \{\mathbf{2}\colon \frac{1}{2}, \mathbf{4}\colon \frac{1}{2}\})$
$\mathbf{3} \mapsto (\mathsf{F}, \{\mathbf{2}\colon \frac{1}{4}, \mathbf{4}\colon \frac{1}{2}, \mathbf{5}\colon \frac{1}{4}\})$
$\mathbf{4} \mapsto (\mathsf{T}, \{\mathbf{4}\colon 1\})$
$\mathbf{5} \mapsto (\mathsf{F}, \{\mathbf{3}\colon \frac{1}{2}, \mathbf{4}\colon \frac{1}{2}\})$</td>
</tr>
</table>

<table>
<tr><th>DFA</th><th>Transition system</th><th>Markov chain</th></tr>
<tr><td>
</td><td>
</td><td>
</td></tr>
<tr><td>$F(X) = \{\mathsf{F}, \mathsf{T}\} \times X \times X$</td><td>$F(X) = \mathcal{P}_{\mathsf{f}}(X)$</td><td>$F(X) = \{\mathsf{F}, \mathsf{T}\} \times \mathcal{D}(X)$</td></tr>
<tr><td>$1 \mapsto (\mathsf{F}, 2, 3)$
$2 \mapsto (\mathsf{F}, 4, 3)$
$3 \mapsto (\mathsf{F}, 5, 3)$
$4 \mapsto (\mathsf{T}, 5, 4)$
$5 \mapsto (\mathsf{T}, 4, 4)$</td><td>$1 \mapsto \{2, 3, 4\}$
$2 \mapsto \{1, 4\}$
$3 \mapsto \{3, 4, 5\}$
$4 \mapsto \{4, 5\}$
$5 \mapsto \{\,\}$</td><td>$1 \mapsto (\mathsf{F}, \{2\colon \frac{1}{3}, 3\colon \frac{2}{3}\})$
$2 \mapsto (\mathsf{F}, \{2\colon \frac{1}{2}, 4\colon \frac{1}{2}\})$
$3 \mapsto (\mathsf{F}, \{2\colon \frac{1}{4}, 4\colon \frac{1}{2}, 5\colon \frac{1}{4}\})$
$4 \mapsto (\mathsf{T}, \{4\colon 1\})$
$5 \mapsto (\mathsf{F}, \{3\colon \frac{1}{2}, 4\colon \frac{1}{2}\})$</td></tr>
<tr><td>$2 \equiv 3, 4 \equiv 5$</td><td>$1 \equiv 2, 3 \equiv 4$</td><td>$2 \equiv 3 \equiv 5$</td></tr>
</table>

What is coalgebraic bisimilarity minimization?

The input:

- a functor $F(X)$ – describes automaton type
- a coalgebra $t : C \to F(C)$ – the automaton

What is coalgebraic bisimilarity minimization?

The input:

- a functor $F(X)$ – describes automaton type
- a coalgebra $t : C \to F(C)$ – the automaton

The output:

- a partition $p : C \to C'$
  – the equivalence classes of bisimilar states
- s.t. $p(x) = p(y) \implies Fp(t(x)) = Fp(t(y))$
- $|C'|$ as small as possible

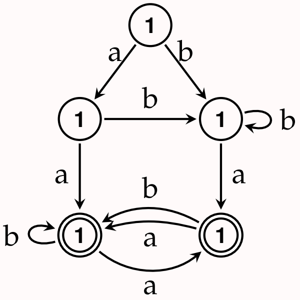
Sketch of our algorithm

- Assume all states are equivalent
- Pick an equivalence class
- Split equivalence class by *signature* (*normalised* outgoing transitions)
- Iterate until convergence

Key points

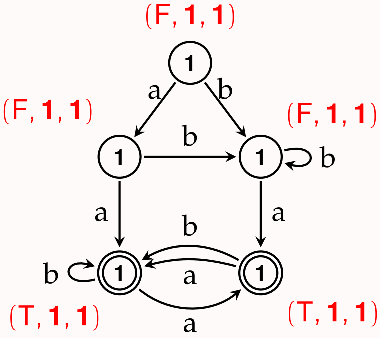
- Only recompute signatures of *changed* states
- Do not loop over *unchanged* states

Our algorithm: Minimizing a DFA



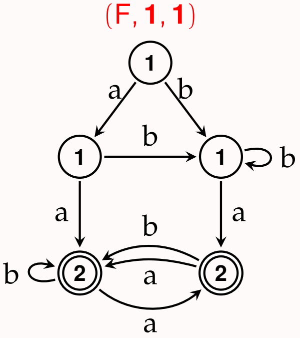
- Set all the state numbers to 1.
- Pick equivalence class
    - Compute missing signatures.
    - Assign new state numbers & Remove signatures from predecessors of changed states.
- Iterate until all states have a signature.

Our algorithm: Minimizing a DFA



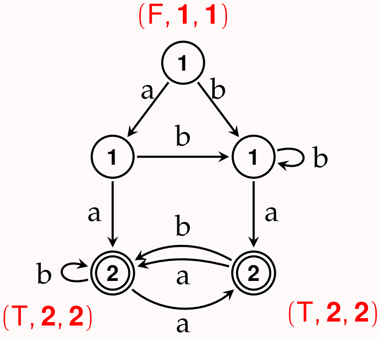
- Set all the state numbers to 1.
- Pick equivalence class
  - Compute missing signatures.
  - Assign new state numbers & Remove signatures from predecessors of changed states.
- Iterate until all states have a signature.

Our algorithm: Minimizing a DFA



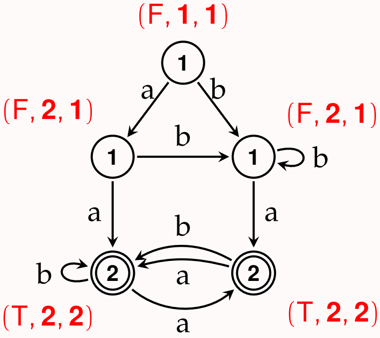
- Set all the state numbers to 1.
- Pick equivalence class
  - Compute missing signatures.
  - Assign new state numbers & Remove signatures from predecessors of changed states.
- Iterate until all states have a signature.

Our algorithm: Minimizing a DFA



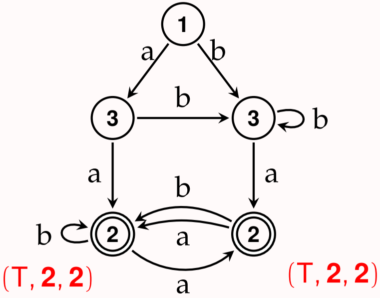
- Set all the state numbers to 1.
- Pick equivalence class
  - Compute missing signatures.
  - Assign new state numbers & Remove signatures from predecessors of changed states.
- Iterate until all states have a signature.

Our algorithm: Minimizing a DFA



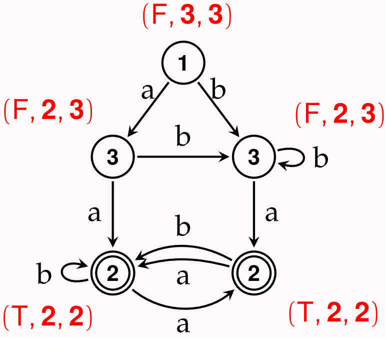
- Set all the state numbers to 1.
- Pick equivalence class
  - Compute missing signatures.
  - Assign new state numbers & Remove signatures from predecessors of changed states.
- Iterate until all states have a signature.

Our algorithm: Minimizing a DFA



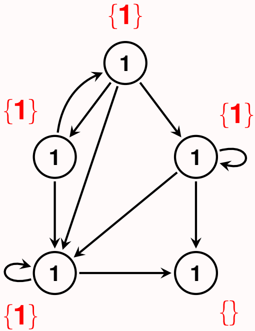
- Set all the state numbers to 1.
- Pick equivalence class
  - Compute missing signatures.
  - Assign new state numbers & Remove signatures from predecessors of changed states.
- Iterate until all states have a signature.

Our algorithm: Minimizing a DFA



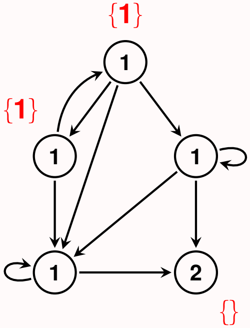
- Set all the state numbers to 1.
- Pick equivalence class
  - Compute missing signatures.
  - Assign new state numbers & Remove signatures from predecessors of changed states.
- Iterate until all states have a signature.

Our algorithm: Minimizing a transition system



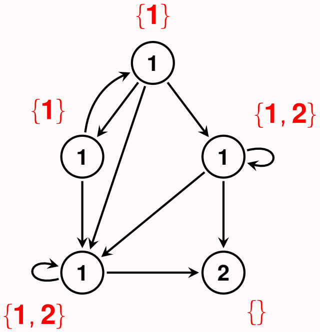
- Set all the state numbers to 1.
- Pick equivalence class
  - Compute missing signatures.
  - Assign new state numbers & Remove signatures from predecessors of changed states.
- Iterate until all states have a signature.

Our algorithm: Minimizing a transition system



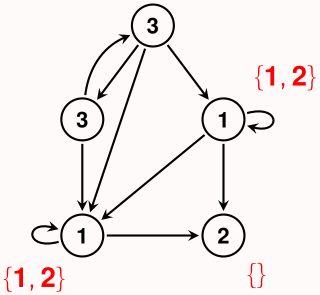
- Set all the state numbers to 1.
- Pick equivalence class
  - Compute missing signatures.
  - Assign new state numbers & Remove signatures from predecessors of changed states.
- Iterate until all states have a signature.

Our algorithm: Minimizing a transition system



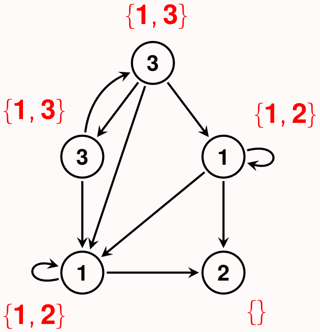
- Set all the state numbers to 1.
- Pick equivalence class
  - Compute missing signatures.
  - Assign new state numbers & Remove signatures from predecessors of changed states.
- Iterate until all states have a signature.

Our algorithm: Minimizing a transition system



- Set all the state numbers to 1.
- Pick equivalence class
  - Compute missing signatures.
  - Assign new state numbers & Remove signatures from predecessors of changed states.
- Iterate until all states have a signature.

Our algorithm: Minimizing a transition system



- Set all the state numbers to 1.
- Pick equivalence class
  - Compute missing signatures.
  - Assign new state numbers & Remove signatures from predecessors of changed states.
- Iterate until all states have a signature.

Caveat

The previous examples are over-simplified, but the real algorithm is not complicated.

- The only complex part is *not looping over the unchanged states*
- See our paper for details

What we need from the automaton

- Ability to (re)compute signatures
- Ability to determine predecessors

What we need from the automaton

- Ability to (re)compute signatures
- Ability to determine predecessors

Complexity: $O(m \log n)$ signature computations

What we need from the automaton

- Ability to (re)compute signatures
- Ability to determine predecessors

Complexity: $O(m \log n)$ signature computations

- How many times does a state's number change?

What we need from the automaton

- Ability to (re)compute signatures
- Ability to determine predecessors

Complexity: $O(m \log n)$ signature computations

- How many times does a state's number change?
  - At most $O(\log n)$ times, if we re-use the old state number for the largest new equivalence class ("Hopcroft's trick")

What we need from the automaton

- Ability to (re)compute signatures
- Ability to determine predecessors

Complexity: $O(m \log n)$ signature computations

- How many times does a state's number change?
  - At most $O(\log n)$ times, if we re-use the old state number for the largest new equivalence class ("Hopcroft's trick")
- How many times does a signature get computed?

What we need from the automaton

- Ability to (re)compute signatures
- Ability to determine predecessors

Complexity: $O(m \log n)$ signature computations

- How many times does a state's number change?
  - At most $O(\log n)$ times, if we re-use the old state number for the largest new equivalence class ("Hopcroft's trick")
- How many times does a signature get computed?
  - At most $O(\log n)$ times per edge

What we need from the automaton

- Ability to (re)compute signatures
- Ability to determine predecessors

Complexity: $O(m \log n)$ signature computations

- How many times does a state's number change?
  - At most $O(\log n)$ times, if we re-use the old state number for the largest new equivalence class ("Hopcroft's trick")
- How many times does a signature get computed?
  - At most $O(\log n)$ times per edge
- At most $O(m \log n)$ signature computations

What we need from the automaton

- Ability to (re)compute signatures
- Ability to determine predecessors

Complexity: $O(m \log n)$ signature computations

- How many times does a state's number change?
  - At most $O(\log n)$ times, if we re-use the old state number for the largest new equivalence class ("Hopcroft's trick")
- How many times does a signature get computed?
  - At most $O(\log n)$ times per edge
- At most $O(m \log n)$ signature computations
  - Total complexity: usually $O(km \log n)$

What we need from the automaton

- Ability to (re)compute signatures
- Ability to determine predecessors

Complexity: $O(m \log n)$ signature computations

- How many times does a state's number change?
  - At most $O(\log n)$ times, if we re-use the old state number for the largest new equivalence class ("Hopcroft's trick")
- How many times does a signature get computed?
  - At most $O(\log n)$ times per edge
- At most $O(m \log n)$ signature computations
  - Total complexity: usually $O(km \log n)$
- What about the complexity of bookkeeping?
  - See paper for n-way partition refinement data structure
  - This is the only complex part of the algorithm

Comparison

	CoPaR	*DCPR*	*mCRL2*	*Boa*
Complexity	$O(m \log n)$	$O(\phi_F n^2)$	$O(m \log n)$	$O(\phi_F m \log n)$
Generality	Zippable	Coalg	LTS+	Coalg
Language	Haskell	Haskell	C++	Rust

benchmark		time (s)			memory (MB)	
type	**n**	***CoPaR***	***DCPR***	***Boa***	***DCPR***	***Boa***
fms	**1639440**	232	84	1.12	514×32	196
	4459455	–	406	4.47	1690×32	582
wlan	**607727**	105	855	0.28	147×32	42
	1632799	–	2960	0.79	379×32	93
wta(W)	**152107**	566	79	0.74	642×32	83
	944250	–	675	11.96	6786×32	1228
wta(Z)	**156913**	438	82	0.48	677×32	92
	1007990	–	645	16.75	5644×32	1325
wta(2)	**154863**	449	160	0.81	621×32	79
	1300000	–	1377	23.35	7092×32	1647

What is the cost of generality?

benchmark		time (s)		memory (MB)	
type	**n**	***mCRL2***	***Boa***	***mCRL2***	***Boa***
cwi	**2416632**	13.9	1.4	1780	249
	7838608	214.2	15.8	5777	814
	33949609	282.2	31.5	16615	2776
vasy	**6020550**	33.8	3.1	2124	520
	11026932	51.6	6.1	2768	619
	12323703	56.9	7.0	3103	734

For ***mCRL2***, we pick its best algorithm and self-reported time.
For ***Boa***, we report wall-clock time.

Conclusion

Minimization can be **generic** and **fast**

Conclusion

Minimization can be **generic** and **fast**

Future

Other notions of equivalence (*e.g.*, branching).
Specialization by monomorphisation.
Integration into Storm (with Sebastian Junges).

(P.S., I'm looking for a postdoc position)