State Space Reduction For Parity Automata

Andreas Tollkötter
Supervisor: Dr. Christof Löding

January 16, 2019

Overview

Goal: reduce the number of states in a given deterministic parity automaton while keeping the recognized language.

Overview

Goal: reduce the number of states in a given deterministic parity automaton while keeping the recognized language.

1. Deterministic Parity Automata
2. Why do we need heuristic reduction?
3. Merger functions as a framework
4. Delayed Simulation
5. Congruence Path Refinement
6. Efficiency

Table Of Contents

ω-automata

ω-words are words of one-sided infinite length:
Σ^ω = functions from $\mathbb{N}$ to Σ

ω-automata are finite transition structures that describe a language $L \subseteq \Sigma^\omega$

Deterministic parity automata (DPA):

- State set Q
- Alphabet Σ
- Transition function $\delta : Q \times \Sigma \to Q$
- Priority function $c : Q \to \mathbb{N}$

An ω-word α starting in a state $q_0 \in Q$ induces a run $q_0 q_1 q_2 \ldots$.
The DPA accepts α iff the **smallest** priority that occurs infinitely often in the sequence $c(q_0)c(q_1)c(q_2)\ldots$ is **even**.

Table Of Contents

Why do we need heuristic reduction?

Goal: Reduce number of states in the automaton to ease run time of follow up algorithms.

Minimization Problem: Given an automaton $\mathcal{A}$, what is the smallest number of states required to recognize the same language as $\mathcal{A}$?

For DFAs: Minimization is solvable in $\mathcal{O}(n \log n)$. [Hopcroft, 1971]
For DPAs: Minimization is NP-hard. [Schewe, 2010]

Moore Minimization

A DPA can be interpreted as a Moore automaton with c being the output function.

Definition.

$p \equiv_M q$ iff $\forall w \in \Sigma^* : c(\delta^*(p, w)) = c(\delta^*(q, w))$.

Moore Minimization

A DPA can be interpreted as a Moore automaton with c being the output function.

Definition.

$p \equiv_M q$ iff $\forall w \in \Sigma^* : c(\delta^*(p, w)) = c(\delta^*(q, w))$.

Theorem.

Deterministic Moore automata can be minimized in log-linear time.

Idea: Build the quotient automaton w.r.t. $\equiv_M$.

The same algorithm can be used to reduce DPAs but will not give minimal DPAs in general.

Table Of Contents

Merger functions

Merger functions μ map from some $D \subseteq 2^Q$ into $2^Q \setminus \{\emptyset\}$.

$M, C \subseteq Q$

$$\mu(\ M\ ) =\ C$$

All states from the **merge set** ...

... can be represented by any single one representative from the **candidate set**.

Merger functions generalize quotient automata

Special case: $\mu(M) = M$.
Remove all states from M except for one (arbitrarily chosen) representative.

For a congruence relation $\sim$, let $\mathfrak{C} \subseteq 2^Q$ be the equivalence classes.
The quotient automaton is defined by state set $\mathfrak{C}$.
This is captured by the merger function $\mu_{\div} : \mathfrak{C} \to 2^Q, \kappa \mapsto \kappa$.

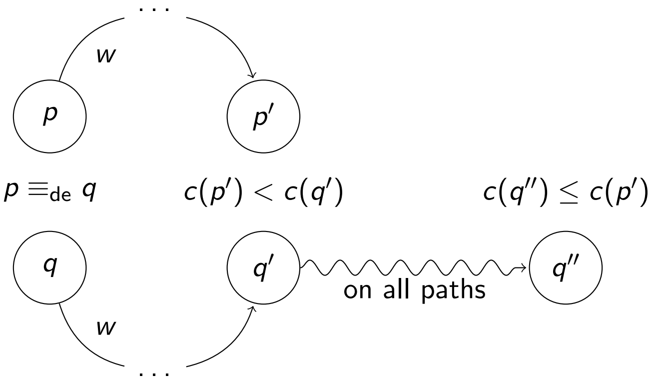
Table Of Contents

Delayed Simulation

Definition.

$p \equiv_{\mathrm{de}} q$ iff for all $w \in \Sigma^*$, every run that starts in $\delta^*(p, w)$ or $\delta^*(q, w)$ eventually sees a priority of at most $\min\{c(\delta^*(p, w)), c(\delta^*(q, w))\}$.

Delayed Simulation



Delayed Simulation

Definition.

Let $\mathfrak{C}_{\mathrm{de}} = \{[q]_{\equiv_{\mathrm{de}}} \mid q \in Q\}$ be the set of $\equiv_{\mathrm{de}}$-equivalence classes.
Define the **delayed simulation merger** as

$$\mu_{\mathrm{de}} : \mathfrak{C}_{\mathrm{de}} \to 2^Q, \kappa \mapsto \{q \in \kappa \mid c(q) = \min c(\kappa)\}.$$

Theorem.

Merging states according to μ_{de} preserves language.

Computing Delayed Simulation

We define a det. Büchi automaton $\mathcal{G}_{\mathrm{de}}$ with states $q^0_{\mathrm{de}}(p, q)$ such that:
$p \equiv_{\mathrm{de}} q$ iff both $L(\mathcal{G}_{\mathrm{de}}, q^0_{\mathrm{de}}(p, q))$ and $L(\mathcal{G}_{\mathrm{de}}, q^0_{\mathrm{de}}(q, p))$ are universal (Σ^ω).

This automaton uses the state set $Q_{\mathrm{de}} = Q \times Q \times (c(Q) \cup \{\checkmark\})$.
Computing states of universal language in a DBA requires linear time.

Theorem.

μ_{de} can be computed in $\mathcal{O}(n^2 k)$.

Delayed Simulation Automaton

$\mathcal{G}_{\text{de}} = (Q_{\text{de}}, \Sigma, \delta_{\text{de}}, F_{\text{de}})$

- States are $Q_{\text{de}} = Q \times Q \times (c(Q) \cup \{\checkmark\})$.
  The first two components are a "simulation" of the original DPA. The third component are the so called "obligations".
- Accepting states are $F_{\text{de}} = Q \times Q \times \{\checkmark\}$.
- Transitions δ_{de}.

$$\begin{aligned}\delta_{\text{de}}((p, q, k), a) = (&\delta(p, a),\\ &\delta(q, a),\\ &\gamma(\quad c(\delta(p, a)), \quad c(\delta(q, a)), \quad k \quad))\end{aligned}$$

Delayed Simulation Automaton: γ

(Actual definition of γ is more complex for some additional properties.)

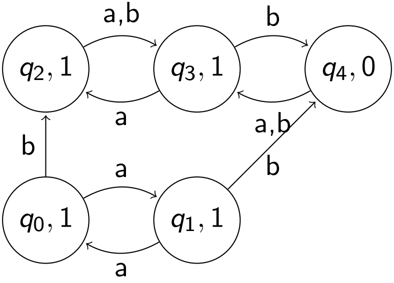
$\gamma : \mathbb{N} \times \mathbb{N} \times (\mathbb{N} \cup \{\checkmark\}) \to \mathbb{N} \cup \{\checkmark\}$

$$\gamma(i, j, \checkmark) = \begin{cases} \checkmark & \text{if } j \leq i \\ i & \text{else} \end{cases}$$

$$\text{for } k \in \mathbb{N} : \quad \gamma(i, j, k) = \begin{cases} \checkmark & \text{if } j \leq \min\{i, k\} \\ \min\{i, k\} & \text{else} \end{cases}$$

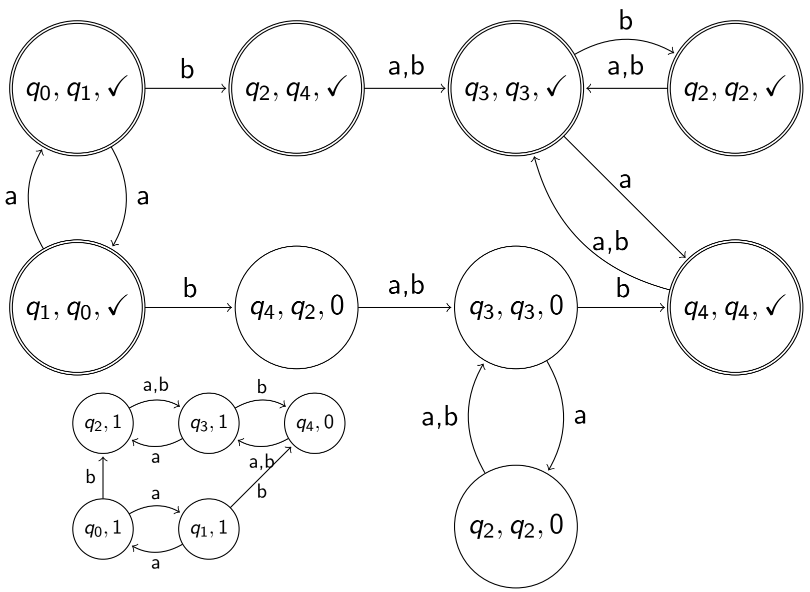
$q^0_{\mathrm{de}}(p, q) = (p, q, \gamma(c(p), c(q), \checkmark))$.

Delayed Simulation Automaton

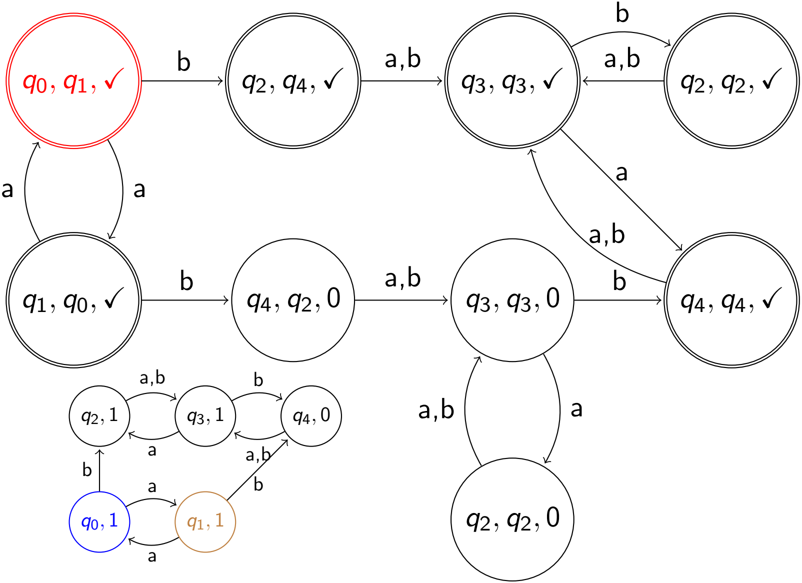


A DPA with 5 states. We want to check whether $q_0 \equiv_{de} q_1$ is true.

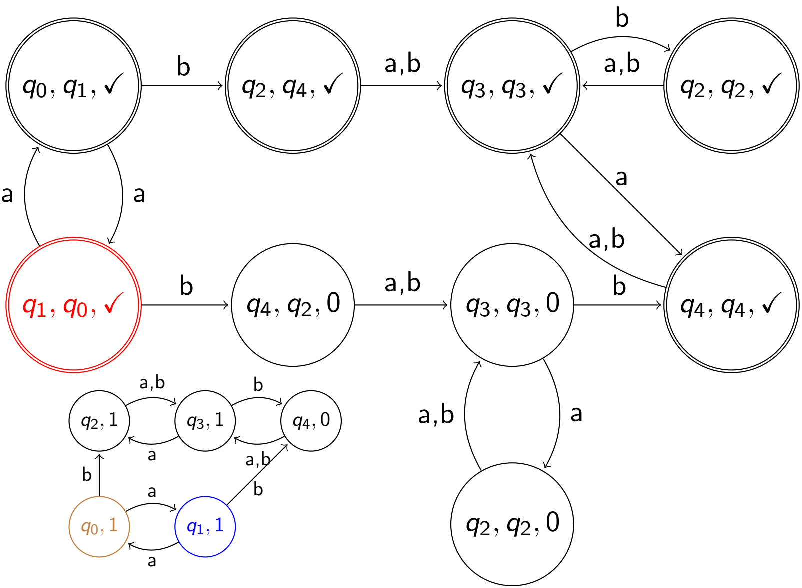
Delayed Simulation Automaton



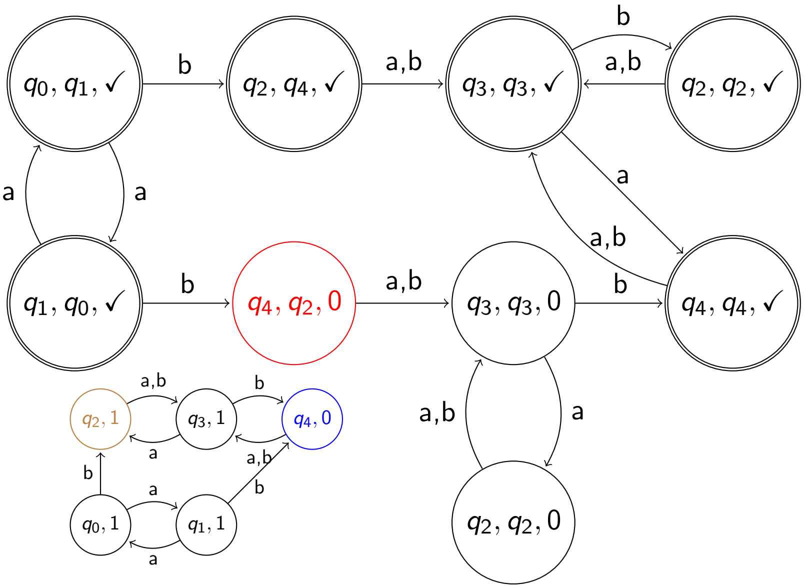
Delayed Simulation Automaton



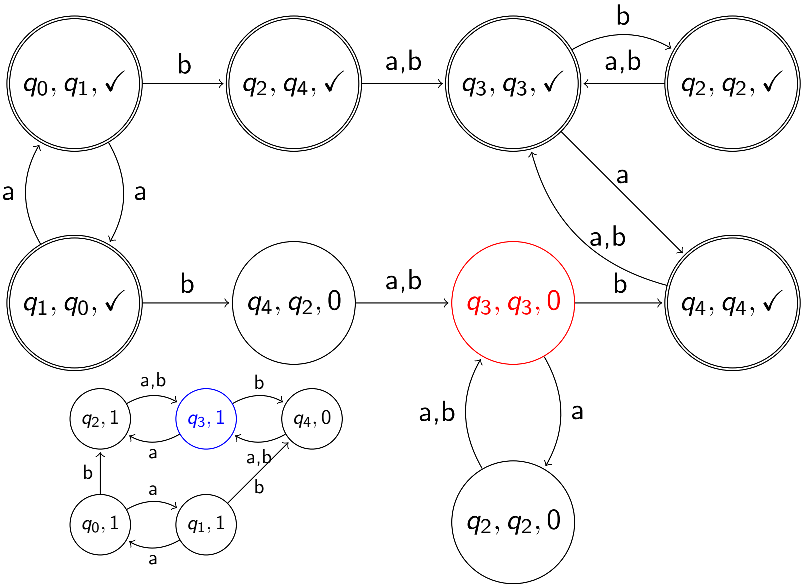
Delayed Simulation Automaton



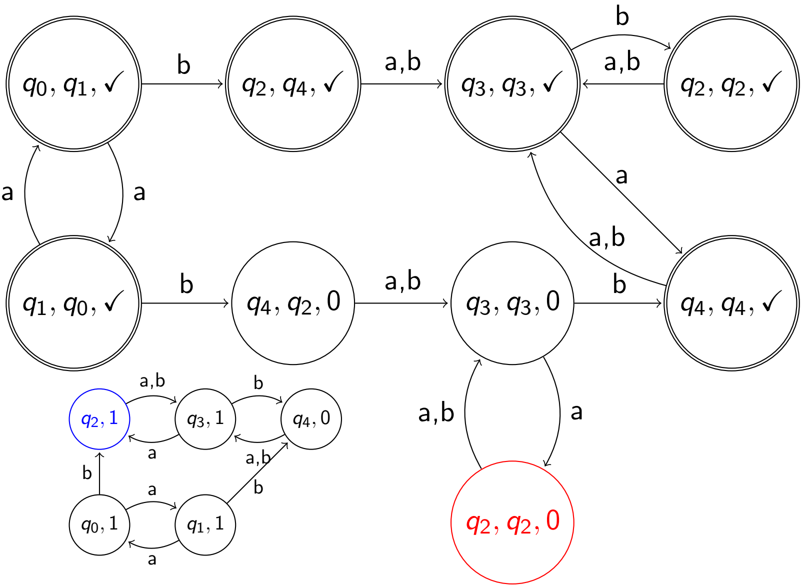
Delayed Simulation Automaton



Delayed Simulation Automaton



Delayed Simulation Automaton



Delayed Simulation Automaton

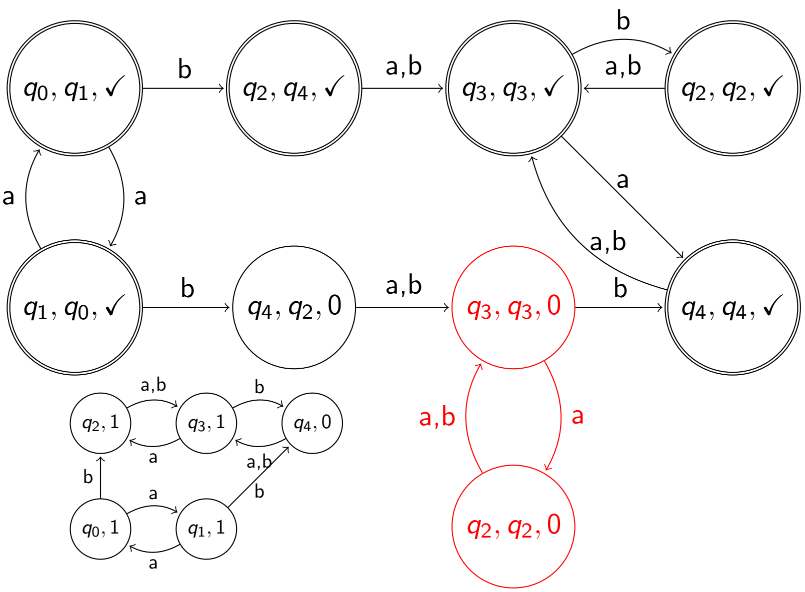


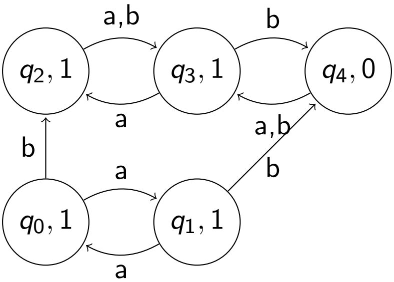
Table Of Contents

Congruence Path Refinement

Idea: take a given relation and refine it until states can be merged.

Definition.

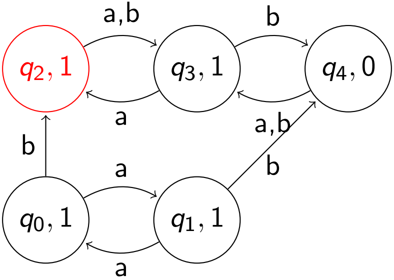
Let $\sim$ be a congruence relation and let $\lambda \subseteq Q$ be an equiv. class of $\sim$. We define $L_{\lambda\hookleftarrow} \subseteq \Sigma^*$ as the set of all words such that the induced run from a state in λ moves back to λ exactly once and ends there.



$\lambda = \{q_2, q_4\}$
Because $\sim$ is a congruence relation, we only need to consider one state.

$L_{\lambda\hookleftarrow} = \{$

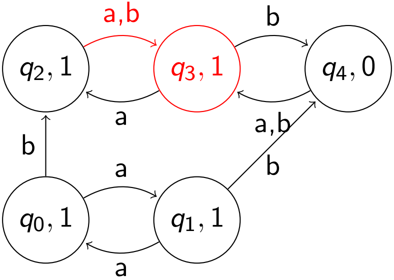
Congruence Path Refinement



$\lambda = \{q_2, q_4\}$
Because $\sim$ is a congruence relation, we only need to consider one state.

$L_{\lambda\hookleftarrow} = \{$

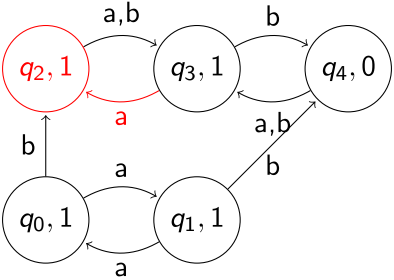
Congruence Path Refinement



$\lambda = \{q_2, q_4\}$

$L_{\lambda\hookleftarrow} = \{$

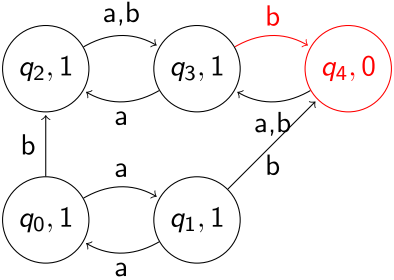
Congruence Path Refinement



$\lambda = \{q_2, q_4\}$

$L_{\lambda\hookleftarrow} = \{aa, ba$

Congruence Path Refinement



$\lambda = \{q_2, q_4\}$

$L_{\lambda\hookleftarrow} = \{aa, ba, ab, bb\}$

Congruence Path Refinement

Definition.

The **path refinement** equivalence $\equiv_{\mathsf{PR}}^{\lambda}$ is the **largest relation** s.t.:
For $p, q \in \lambda$, $p \equiv_{\mathsf{PR}}^{\lambda} q$ if and only if

- $\forall w \in L_{\lambda\hookleftarrow} : \delta^*(p, w) \equiv_{\mathsf{PR}}^{\lambda} \delta^*(q, w)$
- $\forall w \in L_{\lambda\hookleftarrow}$: the smallest priority seen when reading w is the same from p and from q.

Path Refinement Merger

Definition.

Let $\mathfrak{C}^{\lambda}_{\mathsf{PR}} = \{[q]_{\equiv^{\lambda}_{\mathsf{PR}}} \mid q \in Q\}$ be the set of $\equiv^{\lambda}_{\mathsf{PR}}$-equivalence classes. Define the **path refinement merger** as
$\mu^{\lambda}_{\mathsf{PR}} : \mathfrak{C}^{\lambda}_{\mathsf{PR}} \to 2^{Q}, \kappa \mapsto \{q \in \kappa \mid c(q) = \min c(\kappa)\}$.

Theorem.

If all states in λ are pairwise language equivalent, merging states according to $\mu^{\lambda}_{\mathsf{PR}}$ preserves language.

Computing Path Refinement

Definition.

Define the **visit graph** DPA $\mathcal{A}^{\lambda}_{\text{visit}} = (Q^{\lambda}_{\text{visit}}, \Sigma, \delta^{\lambda}_{\text{visit}}, c^{\lambda}_{\text{visit}})$.

- $Q^{\lambda}_{\text{visit}} = Q \times c(Q) \times (c(Q) \cup \{-1\})$
- $\delta^{\lambda}_{\text{visit}}((q, k, k'), a) = \begin{cases} (q', \min\{k, c(q')\}, -1) & \text{if } q' \notin \lambda \\ (q', c(q'), \min\{k, c(q')\}) & \text{if } q' \in \lambda \end{cases}$,
  where $q' = \delta(q, a)$.
- $c^{\lambda}_{\text{visit}}((q, k, k')) = k'$.

The first component "simulates" the original automaton $\mathcal{A}$.
The second component tracks the minimal priority seen on one run from λ to λ.
The third component is required to distinguish the different priorities.

Computing Path Refinement

Definition.

Define the **visit graph** DPA $\mathcal{A}^{\lambda}_{\text{visit}} = (Q^{\lambda}_{\text{visit}}, \Sigma, \delta^{\lambda}_{\text{visit}}, c^{\lambda}_{\text{visit}})$.

- $Q^{\lambda}_{\text{visit}} = Q \times c(Q) \times (c(Q) \cup \{-1\})$
- $\delta^{\lambda}_{\text{visit}}((q,k,k'),a) = \begin{cases} (q', \min\{k, c(q')\}, -1) & \text{if } q' \notin \lambda \\ (q', c(q'), \min\{k, c(q')\}) & \text{if } q' \in \lambda \end{cases}$,
  where $q' = \delta(q,a)$.
- $c^{\lambda}_{\text{visit}}((q,k,k')) = k'$.

The first component "simulates" the original automaton $\mathcal{A}$.
The second component tracks the minimal priority seen on one run from λ to λ.
The third component is required to distinguish the different priorities.

Computing Path Refinement

Definition.

Define the **visit graph** DPA $\mathcal{A}^{\lambda}_{\text{visit}} = (Q^{\lambda}_{\text{visit}}, \Sigma, \delta^{\lambda}_{\text{visit}}, c^{\lambda}_{\text{visit}})$.

- $Q^{\lambda}_{\text{visit}} = Q \times c(Q) \times (c(Q) \cup \{-1\})$
- $\delta^{\lambda}_{\text{visit}}((q, k, k'), a) = \begin{cases} (q', \min\{k, c(q')\}, -1) & \text{if } q' \notin \lambda \\ (q', c(q'), \min\{k, c(q')\}) & \text{if } q' \in \lambda \end{cases}$,
  where $q' = \delta(q, a)$.
- $c^{\lambda}_{\text{visit}}((q, k, k')) = k'$.

The first component "simulates" the original automaton $\mathcal{A}$.
The second component tracks the minimal priority seen on one run from λ to λ.
The third component is required to distinguish the different priorities.

Computing Path Refinement

Definition.

Define the **visit graph** DPA $\mathcal{A}^{\lambda}_{\text{visit}} = (Q^{\lambda}_{\text{visit}}, \Sigma, \delta^{\lambda}_{\text{visit}}, c^{\lambda}_{\text{visit}})$.

- $Q^{\lambda}_{\text{visit}} = Q \times c(Q) \times (c(Q) \cup \{-1\})$
- $\delta^{\lambda}_{\text{visit}}((q, k, k'), a) = \begin{cases} (q', \min\{k, c(q')\}, -1) & \text{if } q' \notin \lambda \\ (q', c(q'), \min\{k, c(q')\}) & \text{if } q' \in \lambda \end{cases}$,
  where $q' = \delta(q, a)$.
- $c^{\lambda}_{\text{visit}}((q, k, k')) = k'$.

The first component "simulates" the original automaton $\mathcal{A}$.
The second component tracks the minimal priority seen on one run from λ to λ.
The third component is required to distinguish the different priorities.

Computing Path Refinement

Moore equivalence in $\mathcal{A}^{\lambda}_{\text{visit}}$ corresponds to path refinement equivalence in $\mathcal{A}$.

Definition.

For $q \in Q$, we have $\iota_q := (q, c(q), \max c(Q)) \in Q^{\lambda}_{\text{visit}}$.

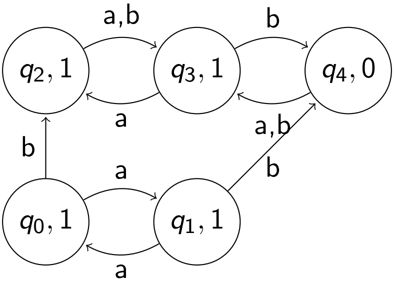
Theorem.

$p \equiv^{\lambda}_{PR} q$ iff $\iota_p \equiv_M \iota_q$.

Theorem.

$\equiv^{\lambda}_{PR}$ can be computed in $\mathcal{O}(k^2 n \log n)$.

Visit Graph



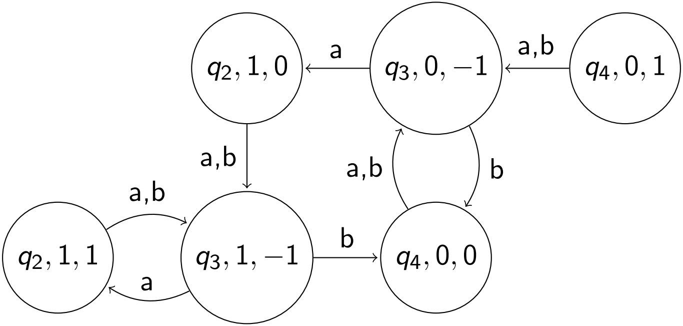
Potential choices for λ are the equivalence classes of $\equiv_L$:
$\{q_0, q_1\}$, $\{q_2, q_4\}$, or $\{q_3\}$.

We take $\lambda = \{q_2, q_4\}$ and ask if $q_2 \equiv^{\lambda}_{\mathrm{PR}} q_4$ is true.

Visit Graph

$\mathcal{A}_{\text{visit}}^{\{q_2,q_4\}}$

$\iota_{q_2} = (q_2, 1, 1)$ $\iota_{q_4} = (q_4, 0, 1)$ Question: $\iota_{q_2} \equiv_M \iota_{q_4}$?

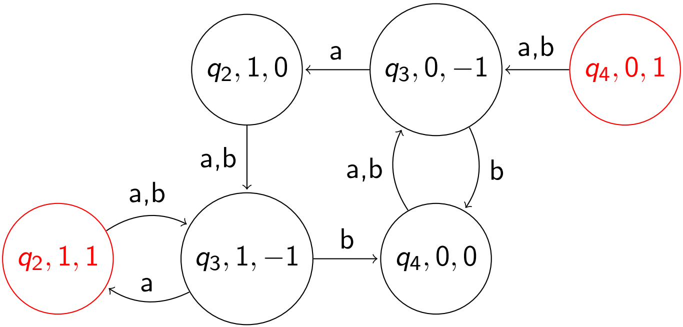


(Reminder: the third component defines the color of a state)

Visit Graph

$\mathcal{A}_{\text{visit}}^{\{q_2, q_4\}}$

$\iota_{q_2} = (q_2, 1, 1)$ $\iota_{q_4} = (q_4, 0, 1)$ Question: $\iota_{q_2} \equiv_M \iota_{q_4}$?

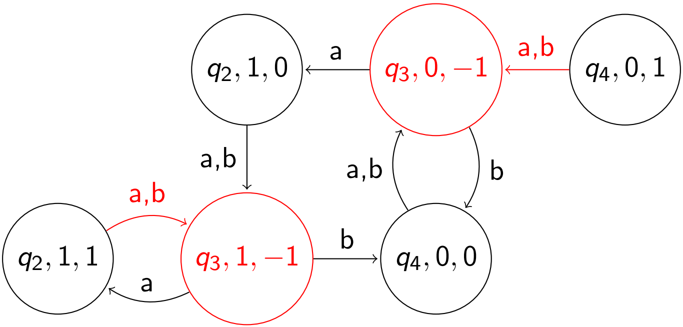


(Reminder: the third component defines the color of a state)

Visit Graph

$\mathcal{A}_{\text{visit}}^{\{q_2,q_4\}}$

$\iota_{q_2} = (q_2, 1, 1)$ $\iota_{q_4} = (q_4, 0, 1)$ Question: $\iota_{q_2} \equiv_M \iota_{q_4}$?



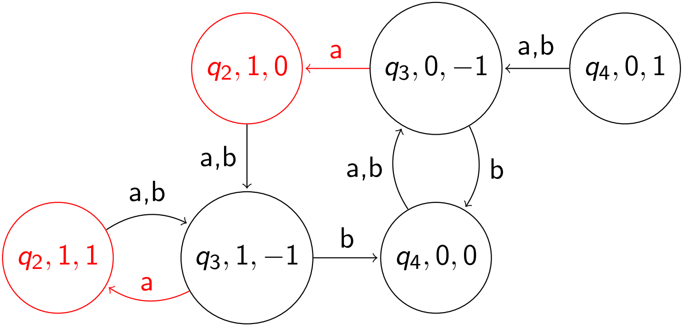
(Reminder: the third component defines the color of a state)

Visit Graph

$\mathcal{A}_{\text{visit}}^{\{q_2,q_4\}}$

$\iota_{q_2} = (q_2, 1, 1) \qquad \iota_{q_4} = (q_4, 0, 1)$

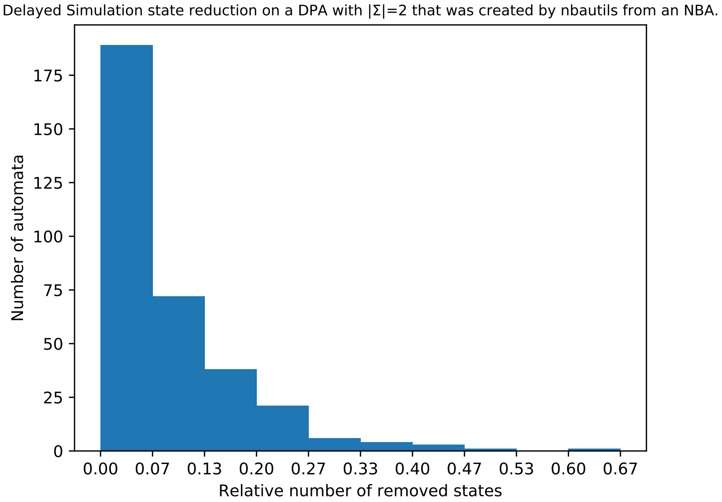
Question: $\iota_{q_2} \equiv_M \iota_{q_4}$?



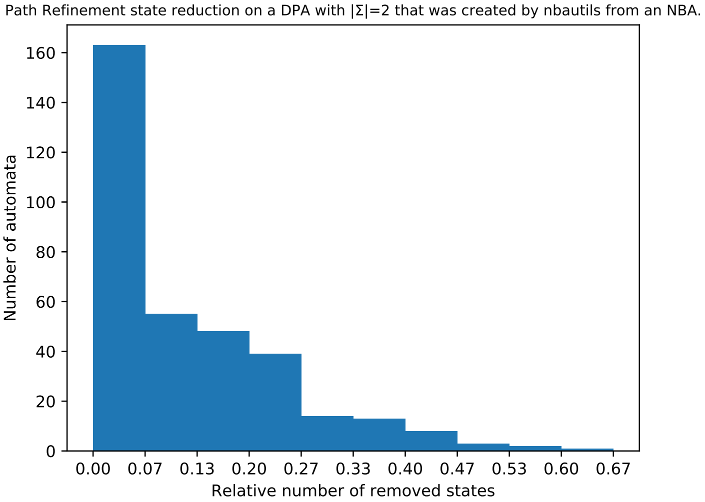
q_2 and q_4 are not PR-equivalent.

Table Of Contents

Delayed Simulation



Path Refinement



Summary

- Moore
- Skip
- Delayed Simulation
- Iterated Moore
- Path Refinement
- Threshold Moore
- LSF

$\Rightarrow$ Merger function $\Rightarrow$

- Representative merge
- Schewe merge

Bonchi, F. and Pous, D. (2013).
Checking nfa equivalence with bisimulations up to congruence.
In *Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '13, pages 457–468, New York, NY, USA. ACM.

Büchi, J. R. (1966).
On a decision method in restricted second order arithmetic.

Carton, O. and Maceiras, R. (1999).
Computing the rabin index of a parity automaton.
RAIRO - Theoretical Informatics and Applications - Informatique Théorique et Applications, 33(6):495–505.

Champarnaud, J.-M. and Coulon, F. (2004).
Nfa reduction algorithms by means of regular inequalities.
Theoretical Computer Science, 327(3):241 – 253.
Developments in Language Theory.

Duret-Lutz, A., Lewkowicz, A., Fauchille, A., Michaud, T., Renault, E., and Xu, L. (2016).
Spot 2.0 — a framework for LTL and ω-automata manipulation.
In *Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis (ATVA'16)*, volume 9938 of *Lecture Notes in Computer Science*, pages 122–129. Springer.

Etessami, K., Wilke, T., and Schuller, R. A. (2001).
Fair simulation relations, parity games, and state space reduction for büchi automata.
In *Automata, Languages and Programming*, pages 694–707, Berlin, Heidelberg. Springer Berlin Heidelberg.

Fritz, C. and Wilke, T. (2005).
Simulation relations for alternating büchi automata.
Theor. Comput. Sci., 338(1-3):275–314.

Henzinger, M. R. and Telle, J. A. (1996).
Faster algorithms for the nonemptiness of streett automata and for communication protocol pruning.
In *Algorithm Theory — SWAT'96*, pages 16–27, Berlin, Heidelberg. Springer Berlin Heidelberg.

Hopcroft, J. (1971).
An n log n algorithm for minimizing states in a finite automaton.
An N Log N Algorithm for Minimizing States in A Finite Automaton, page 15.

Jiang, T. and Ravikumar, B. (1991).
Minimal nfa problems are hard.
In *Automata, Languages and Programming*, pages 629–640, Berlin, Heidelberg. Springer Berlin Heidelberg.

Mayr, R. and Clemente, L. (2012).
Advanced automata minimization.
In *POPL 2013*.

McCulloch, W. S. and Pitts, W. (1990).
A logical calculus of the ideas immanent in nervous activity. 1943.
Bulletin of mathematical biology, 52 1-2:99–115; discussion 73–97.

Michel, M. (1988).
Complementation is much more difficult with automata on infinite words.

Moore, E. F. (1956).
Gedanken-experiments on sequential machines.
In Shannon, C. and McCarthy, J., editors, *Automata Studies*, pages 129–153. Princeton University Press, Princeton, NJ.

Mostowski, A. (1991).
Hierarchies of weak automata and weak monadic formulas.
Theoretical Computer Science, 83(2):323 – 335.

Pirogov, A. (2018).
nbautils.
`https://github.com/apirogov/nbautils`.

Piterman, N. (2007).
From nondeterministic büchi and streett automata to deterministic parity automata.
Logical Methods in Computer Science, 3(3).

Rabin, M. O. and Scott, D. (1959).
Finite automata and their decision problems.
IBM Journal of Research and Development, 3:114–125.

Schewe, S. (2009).
Tighter bounds for the determinisation of büchi automata.
In de Alfaro, L., editor, *Foundations of Software Science and Computational Structures*, pages 167–181, Berlin, Heidelberg. Springer Berlin Heidelberg.

Schewe, S. (2010).
Beyond Hyper-Minimisation—Minimising DBAs and DPAs is NP-Complete.
In Lodaya, K. and Mahajan, M., editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010)*, volume 8 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 400–411, Dagstuhl, Germany. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

Sharir, M. (1981).
A strong-connectivity algorithm and its applications in data flow analysis.
Computers & Mathematics with Applications, 7(1):67 – 72.

Tarski, A. (1955).
A lattice-theoretical fixpoint theorem and its applications.
Pacific J. Math., 5(2):285–309.

Thomas, W. (1990).
Handbook of theoretical computer science (vol. b).
chapter Automata on Infinite Objects, pages 133–191. MIT Press, Cambridge, MA, USA.

Thomas, W. (1997).
Handbook of formal languages, vol. 3.
chapter Languages, Automata, and Logic, pages 389–455. Springer-Verlag New York, Inc., New York, NY, USA.

Tollkötter, A. (2018).
Master thesis.
https://github.com/Wurstinator/master-thesis.