State Space Reduction For Parity Automata

Andreas Tollkötter
Supervisor: Dr. Christof Löding

January 10, 2019

Overview

We establish the notion of **merger functions**. Using that definition, we present three of our newly developed heuristic techniques to reduce the number of states in deterministic parity automata.

Overview

We establish the notion of **merger functions**. Using that definition, we present three of our newly developed heuristic techniques to reduce the number of states in deterministic parity automata.

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. Labeled SCC Filter

Table Of Contents

ω-automata

ω-words are words of one-sided infinite length: $\alpha \in \Sigma^\omega \Leftrightarrow \alpha : \mathbb{N} \to \Sigma$
a^ω, $aa(ba)^\omega$, $(abc)^\omega$

ω-automata are finite transition structures that describe a language
$L(\mathcal{A}) \subseteq \Sigma^\omega$
$\{a^n b^\omega \mid n \in \mathbb{N}\}$

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 $q_0 q_1 q_2 \dots$. The DPA accepts α iff the **smallest** priority that occurs infinitely often in the sequence $c(q_0)c(q_1)c(q_2)\dots$ 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 deterministic finite automata (on finite words): Minimization is solvable in $\mathcal{O}(n \log n)$.

For DPAs: Minimization is NP-hard. []

Moore Minimization

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

Theorem.

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

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

Definition.

Let $\mathcal{A} = (Q, \Sigma, \delta, c)$ be a DPA. A **merger function** is a function $\mu : D \to 2^Q \setminus \{\emptyset\}$ such that

- all sets in D are pairwise disjoint
- for all $X \in D$, $\mu(X) \cap (U \setminus X) = \emptyset$, where $U = \bigcup D$

$\mu(M) = C$
Merge all states in $M \subseteq Q$ into any one representative of $C \subseteq Q$.

For a congruence relation $\sim$, the quotient automaton is defined by state set $Q_\sim = \{[q]_\sim \mid q \in Q\}$.
This is captured by the merger function $\mu_\div : Q_\sim \to 2^Q, \kappa \mapsto \kappa$.

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 $\min\{c(\delta^*(p, w)), c(\delta^*(q, w))\}$.

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 $\mu_{\mathrm{de}} : \mathfrak{C}_{\mathrm{de}} \to 2^Q, \kappa \mapsto c^{-1}(\min c(\kappa))$.

Theorem.

Merging states according to μ_{de} preserves language.

Computing Delayed Simulation

We define a deterministic Büchi automaton $\mathcal{G}_{\mathrm{de}}$ 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, i.e. 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.

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

Delayed Simulation Automaton

$\mathcal{G}_{\mathrm{de}} = (Q_{\mathrm{de}}, \Sigma, \delta_{\mathrm{de}}, F_{\mathrm{de}})$
States are $Q_{\mathrm{de}} = Q \times Q \times (c(Q) \cup \{\checkmark\})$.
The first two components are a "simulation" of the original DPA. The third component are the so called "obligations".
Transitions δ_{de}.
The first two components mimic the transitions of $\mathcal{A}$. The third component is defined by $\gamma : Q_{\mathrm{de}} \times \Sigma \to c(Q) \cup \{\checkmark\}$. (next slide)
Accepting states are $F_{\mathrm{de}} = Q \times Q \times \{\checkmark\}$.

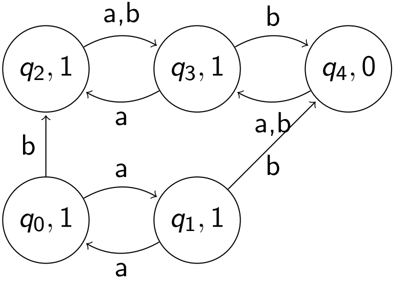
Delayed Simulation Automaton: γ

Let $0 \leq_{\checkmark} 1 \leq_{\checkmark} 2 \leq_{\checkmark} \cdots \leq_{\checkmark} \checkmark$.
For $p, q \in Q$, $k \in c(Q) \cup \{\checkmark\}$, $a \in \Sigma$, set
$\gamma((p, q, k), a) = \gamma'(\delta^*(p, a), \delta^*(q, a), k)$, where γ' is defined as follows:
If any of the following is true, then $\gamma'(i, j, k) = \checkmark$.

- i is odd, j is even, and $i \leq_{\checkmark} k$
- i is odd, j is even, and $j \leq_{\checkmark} k$
- i is odd, j is odd, $j \geq i$, and $i \leq_{\checkmark} k$
- i is even, j is even, $j \leq i$, and $j \leq_{\checkmark} k$

Otherwise, $\gamma'(i, j, k) = min_{\leq_{\checkmark}}\{i, j, k\}$.
$q^0_{\mathrm{de}}(p, q) = (p, q, \gamma'(c(p), c(q), \checkmark))$.

Delayed Simulation Automaton



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

Delayed Simulation Automaton

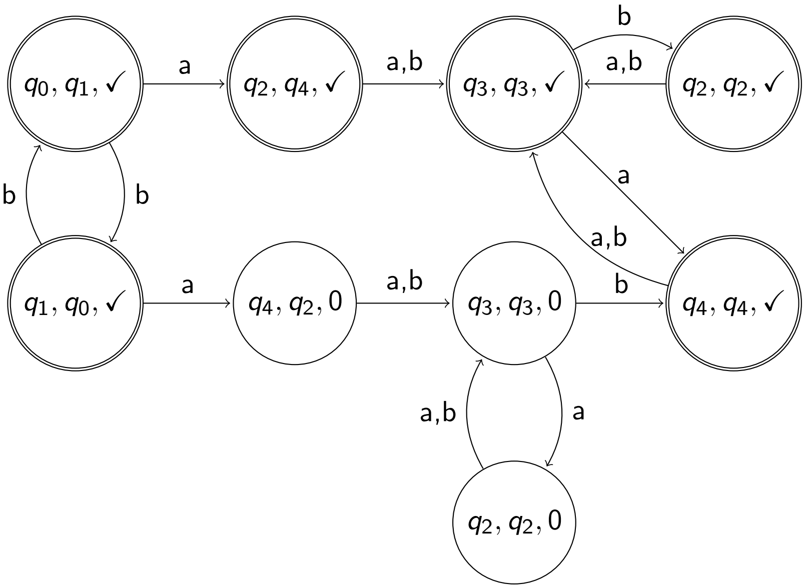


Table Of Contents

Congruence Path Refinement

Definition.

Let $\sim$ be a congruence relation and let $\lambda \subseteq Q$ be an equivalence class of $\sim$. The **path refinement** equivalence $\equiv_{\mathsf{PR}}^{\lambda}$ is the smallest relation