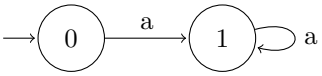

Figure 1: Example automaton in which the states could be merged but delayed simulation separates them.

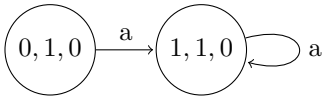

Figure 2: Example automaton in which the states could be merged but delayed simulation separates them.

0.0.1 Alternative computation

As we have seen, using delayed simulation to build a quotient automaton delivers good results in the number of removed states. The downside is the computation time which is much higher than that of our approach in section **??**. In the following we will consider alternations to the delayed simulation algorithm with the goal to increase the number of removed states or to reduce computation time.

Resetting obligations

In the delayed simulation automaton, "obligations" correspond to good priorities that the first state has accumulated or bad priorities that the second state has accumulated and the need for the respective other state to compensate in some way. The intuitive idea behind this concept is that an obligation that cannot be compensated, stands for an infinite run in which the acceptance differs between the two states that are being compared. The issue with the original definition is that obligations carry over, even if they can only be caused finitely often. This is demonstrated in figure 1; the two states could be merged into one, but they are not $\equiv_{\mathrm{de}}$-equivalent as can be seen in the delayed simulation automaton in figure 2.

As a solution to this, we propose a simple change to the definition of the automaton which resets the obligations every time, either state moves to a new SCC.

Definition 0.0.1. Let $\mathcal{A} = (Q, \Sigma, q_0, \delta, c)$ be a DPA. We define the *delayed simulation automaton with SCC resets* $\mathcal{A}_{\mathrm{deR}}(p, q) = (Q_{\mathrm{de}}, \Sigma, (p, q, \gamma(c(p), c(q), \checkmark), \delta_{\mathrm{deR}}, F_{\mathrm{de}})$ with $\delta_{\mathrm{deR}}((p, q, k), a) = \delta_{\mathrm{de}}((p, q, \mathrm{reset}(p, q, k, a)), a)$. Except for the addition of the reset function, this automaton is the same as $\mathcal{A}_{\mathrm{de}}$.

If p and $\delta(p, a)$ lie in the same SCC, as well as q and $\delta(q, a)$, then we simply set $\mathrm{reset}(p, q, k, a) = k$. Otherwise, i.e. if any state changes its SCC, the reset comes into play and we set $\mathrm{reset}(p, q, k, a) = \checkmark$.

We write $p \leq_{\mathrm{deR}} q$ if $L(\mathcal{A}_{\mathrm{deR}}(p, q)) = \Sigma^\omega$. If also $q \leq_{\mathrm{deR}} p$ holds, we write $p \equiv_{\mathrm{deR}} q$.

As the definition is so similar to the original delayed simulation, most results that we have already proven translate directly to the new relation.

Theorem 0.0.1. *$\equiv_{deR}$ is a congruence relation.*

Lemma 0.0.2. *Let $\mathcal{A}$ be a DPA with two states p and q. If $p \leq_{de} q$, then $p \leq_{deR} q$.*

The real difference comes up when looking at theorem **??**. If we consider again the example given in figure 1, if that theorem would hold the same for $\equiv_{\mathrm{deR}}$ we could assign both states the priority 0, which would then make the automaton accept every word. This is obviously not the same as the original automaton, so this statement deserves some additional inspection.

Lemma 0.0.3. *Let $\mathcal{A}$ be a DPA with states $p, q \in Q$. Then $p \preceq_{de} q$ if and only if the following property holds for all $w \in \Sigma^*$:*

Let $p' = \delta^*(p, w)$ and $q' = \delta^*(q, w)$. If $c(p')$ is even and $c(p') < c(q')$, then every path from q' eventually reaches a priority at most $c(p')$. On the other hand, if $c(q')$ is odd and $c(q') < c(p')$, then every path from p' eventually reaches a priority at most $c(q')$.

Proof. **If** We show the contrapositive. Let $p \not\preceq_{\mathrm{de}} q$, so there is a word $\alpha \notin L(\mathcal{A}_{\mathrm{de}}(p, q))$ with the run of $\mathcal{A}_{\mathrm{de}}(p, q)$ being $(p_0, q_0, k_0)(p_1, q_1, k_1)\dots$. By Lemma **??**, there is a position n such that k_i does not change anymore for $i \geq n$. We define $w = \alpha[0, n_0]$ and $\beta = \alpha[n_0 + 1,]$ (so $\alpha = w\beta$) and claim that w is a valid counterexample for the right-side property.

The first case is: $c(p') < c(q')$ and $c(p')$ is even.
Reading β from q' induces a run that never visits a priority less or equal to $c(p')$: Let $u \sqsubset \beta$ and assume that $c(\delta^*(q', u)) \leq c(p')$. By choice of n, we know that $k_{|wu|} = k_{|wu|+1} \neq \checkmark$. This can only happen if the "else" case of γ is hit, meaning that $k_{|wu|+1} = \min\{k_{|wu|}, c(\delta^*(p', u)), c(\delta^*(q', u))\}$. Specifically, $c(\delta^*(q', u)) \geq k_{|wu|}$. By choice of w we also have $k_{|wu|} = k_{|w|} \leq c(p')$, so $c(\delta^*(q', u)) = c(p')$.

This, however, means that $c(\delta^*(q', u))$ is even, $c(\delta^*(q', u)) \leq_{\checkmark} k_{|wu|}$, and $c(\delta^*(q', u)) \preceq_p c(p')$ and thus $k_{|wu|+1} = \checkmark$ which is a contradiction.

The second case, $c(q') < c(p')$ and $c(q')$ is odd, works almost identically so we omit the proof here.

Only If Again we show the contrapositive: There is a $w \in \Sigma^*$ such that the right-side property is violated. Let this w now be chosen among all these words such that $\min\{c(p'), c(q')\}$ becomes minimal. We now show that $p \not\preceq_{\mathrm{de}} q$.

The first case is: $c(p') < c(q')$ and $c(p')$ is even.
Let $\beta \in \Sigma^\omega$ be a word such that the respective run from q' only sees priorities strictly greater than $c(p')$. Let $(p_0, q_0, k_0)(p_1, q_1, k_1)\dots$ be the run of $\mathcal{A}_{\mathrm{de}}(p, q)$ on $\alpha = w\beta$. We claim that $k_i \neq \checkmark$ for all $i > |w|$. If that is true, then the run is rejecting and $\alpha \notin L(\mathcal{A}_{\mathrm{de}}(p, q))$.

Assume towards a contradiction that k_i does become $\checkmark$ again at some point. Let $j \geq |w|$ be the minimal position with $k_{j+1} = \checkmark$. Then by definition of γ, $c(q_{j+1}) \leq k_j$ is even or $c(p_{j+1}) \leq k_j$ is odd. In the former case, we would have a contradiction to the choice of β. In the latter case, we would have a contradiction to the choice of w as a word with minimal priority at $c(p')$: since $c(p')$ is even, $c(p_{j+1}) < c(p')$ and from q_{j+1} there is a run that never reaches a smaller priority. Hence, $w \cdot \beta[0, j - |w|]$ would have been our choice for w instead.

The second case, $c(q') < c(p')$ and $c(q')$ is odd, works almost identically so we omit the proof here. □

While this characterization of $\preceq_{\mathrm{de}}$ seems arbitrary, it allows for an easier definition of $\equiv_{\mathrm{de}}$ as is seen in the following statement.

Corollary 0.0.4. *Let $\mathcal{A}$ be a DPA with states $p, q \in Q$. Then $p \equiv_{\mathrm{de}} q$ if and only if the following holds for all words $w \in \Sigma^*$:*

Let $p' = \delta^*(p, w)$ and $q' = \delta^*(q, w)$. Every run that starts in p' or q' eventually sees a priority less than or equal to $\min\{c(p'), c(q')\}$.

This intermediate result now easily gives us the following relation of delayed simulation and Moore-equivalence.

Definition 0.0.2. Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, c)$ be a parity automaton. We call c *normalized* if for every state $q \in Q$ that does not lie in a trivial SCC and all priorities $k \leq c(q)$, there is a path from q to q such that the lowest priority visited is k.

Lemma 0.0.5. *Let $\mathcal{A}$ be a DPA with a normalized priority function and let p and q be states that do not lie in trivial SCCs. Then $p \equiv_{\text{de}} q$ if and only if $p \sim_M q$.*

Proof. The "if"-implication was shown in **??**. For the other direction, let $p \not\sim_M q$, so there is a word $w \in \Sigma^*$ such that $c(p') \neq c(q')$, where $p' = \delta^*(p, w)$ and $q' = \delta^*(q, w)$. Without loss of generality, assume $c(p') < c(q')$.

As c is normalized, there is a word u such that q' reaches again q' via u and sees only priorities greater or equal to $c(q')$. That means that on the path that is obtained from q' by reading u^ω, the priority $c(p')$ is never visited. By corollary 0.0.4, that means $p \not\equiv_{\text{de}} q$. □

Computing a normalized priority function

If we can assure that our priority function is normalized, Moore-equivalence is a nice approximation of delayed simulation-equivalence. In fact, [] provides an algorithm that is suited for these needs. We will briefly reiterate that algorithm here, as the original paper puts the focus on a different aspect.

Lemma 0.0.6. *The* Normalize *function in algorithm 1 returns a normalized priority function for $\mathcal{A}$ that does not change the language.*

Proof. The fact that the language does not change after running Normalize is part of the original paper.

To see that c_N is normalized, assume towards a contradiction that it is not. Let c □

Lemma 0.0.7. *The* Normalize *function in algorithm 1 runs in $\mathcal{O}(nk)$, where n is the number of states and k is the number of priorities.*

Alternative delayed simulation

Theorem 0.0.8. *Let $\mathcal{A} = (Q, \Sigma, q_0, \delta, c)$ be a DPA s.t. c is normalized and $\mathcal{A}$ contains no unreachable states. Let $c' =$ DelayedSimulation$(\mathcal{A})$. Then The delayed simulation-equivalence of $\mathcal{A}$ is the same as the Moore-equivalence of $(Q, \Sigma, q_0, \delta, c')$.*

Proof. Let $s_0, \ldots, s_m$ be the sequence of SCCs as they are chosen in line 5 of the algorithm and let $\mathcal{B}_0, \ldots, \mathcal{B}_{m+1}$ be the sequence of automata that are set in lines 3 and 6. The computation of $\equiv_{\text{de}}$ and $\sim_M$ is independent of the initial state of an automaton and the state set and transition function of $\mathcal{A}$ and $\mathcal{B}_{m+1}$ are the same. It thus suffices to show that for every $0 \leq i \leq m+1$, the Moore-equivalence of $\mathcal{B}_i$ is the same as the delayed-simulation equivalence of $\mathcal{A} \restriction_{Q(\mathcal{B}_i)}$.

For $i = 0$, this is clear, as $Q(\mathcal{B}_0) = \emptyset$. □

Algorithm 1 Normalizing the priority function of a DPA.

1: **function** NORMALIZE($\mathcal{A}$)
Ensure: c_N is normalized in $\mathcal{A}$.
2: $c_D \leftarrow$MAKEDENSE($\mathcal{A}$)
3: $c_N : Q \to \mathbb{N}, q \mapsto c_D(q)$
4: $Q' \leftarrow Q$
5: $k \leftarrow 0$
6: **while** $Q' \neq \emptyset$ **do**
7: $Q' \leftarrow Q' \setminus c_D^{-1}(k)$
8: $S \leftarrow$ Trivial SCCs of $\mathcal{A} \restriction_{Q'}$
9: **for** $q \in S$ **do**
10: $c_N(q) \leftarrow k$
11: **end for**
12: $Q' \leftarrow Q' \setminus S$
13: $k \leftarrow k + 1$
14: **end while**
15: **return** c_N
16: **end function**

17: **function** MAKEDENSE($\mathcal{A}$)
Ensure: For all $0 \leq i \leq \max c(Q)$, $c_D^{-1}(i) \neq \emptyset$.
18: $c_D : Q \to \mathbb{N}, q \mapsto c(q)$
19: **for** $i \leftarrow 0$ to $\max c(Q)$ **do**
20: **while** $c_D^{-1}(i+1) = \emptyset$ **do**
21: **for** $q \in Q : c_D(q) > i$ **do**
22: $c_D(q) \leftarrow c_D(q) - 2$
23: **end for**
24: **end while**
25: **end for**
26: **return** c_D
27: **end function**

Algorithm 2 Compute $\equiv_{\text{de}}$ of a DPA $\mathcal{A}$.

function DELAYEDSIMULATION($\mathcal{A}$)
  $\mathcal{S} \leftarrow$ compute SCCs of $\mathcal{A}$
  $\mathcal{B} \leftarrow (\emptyset, \Sigma, 0, \emptyset, \emptyset)$
  while $\mathcal{S} \neq \emptyset$ **do**
    $s \in \mathcal{S}$ s.t. s can reach no other SCC in $\mathcal{S}$
    $\mathcal{B} \leftarrow$ ADDSCC($\mathcal{A}$, $\mathcal{B}$, s)
    $\mathcal{S} \leftarrow \mathcal{S} \setminus s$
  end while
  return $c_{\mathcal{B}}$
end function

function ADDSCC($\mathcal{A}$, $\mathcal{B}$, s)
  $q_0 \in s$ arbitrary
  Add s to $\mathcal{B}$: $\mathcal{B} \leftarrow (Q \cup s, \Sigma, q_0, \delta_{\mathcal{B}} \cup \delta_{\mathcal{A}} \restriction_{s \times \Sigma}, c_{\mathcal{B}} \cup c_{\mathcal{A}} \restriction_s)$
  $\sim_M \leftarrow$ Moore-equivalence of $\mathcal{B}$
  $\mathcal{C} \leftarrow \mathcal{B} /_{\sim_M}$
  if q_0 is a trivial SCC in $\mathcal{A}$ and is not $\sim_M$-equivalent to any other states **then**
    if There is a state $p \in \mathcal{C}$ such that $\forall a : \delta_{\mathcal{B}}(q_0, a) = \delta_{\mathcal{B}}(p, a)$ **then**
      $c_{\mathcal{B}}(q_0) \leftarrow c_{\mathcal{C}}(p)$
    end if
  end if
  return $\mathcal{B}$
end function