0.1 Introduction

Finite automata are a long established computation model that dates back to sources such as [11] and [14]. A known problem for finite automata is state space reduction, referring to the search of a language-equivalent automaton which uses fewer states than the original object. For deterministic finite automata (DFA), not just reduction but minimization was solved in [8]. Regarding nondeterministic finite automata (NFA), [9] proved the PSPACE-completeness of the minimization problem, which is why reduction algorithms such as [4] and [1] are a popular alternative.

In his prominent work [2], Büchi introduced the model of Büchi automata (BA) as an extension of finite automata to read words of one-sided infinite length. As these ω-automata tend to have higher levels of complexity in comparison to standard finite automata, the potential gain of state space reduction is even greater. Similar to NFAs, exact minimization for deterministic Büchi automata was shown to be NP-complete in [15] and spawned heuristic approaches such as [15], [10], or [5].

As [17] displays, deterministic Büchi automata are a strictly weaker model than nondeterministic Büchi automata. It is therefore interesting to consider different models of ω-automata in which determinism is possible while maintaining enough power to describe all ω-regular languages. Parity automata (PA) are one such model, a mixture of Büchi automata and Moore automata ([12]), that use a parity function rather than the usual acceptance set. [13] showed that deterministic parity automata are in fact sufficient to recognize all ω-regular languages. As for DBAs, the exact minimization problem for DPAs is NP-complete ([15]).

Our goal in this publication is to develop new algorithms for state space reduction of DPAs, partially adapted from existing algorithms for Büchi or Moore automata. We perform theoretical analysis of the algorithms in the form of proofs of correctness and analysis of run time complexity, as well as practical implementation of the algorithms in code to provide empirical data for or against their actual efficiency.

Chapter 1

Theory

1.1 General Results

We first use this section to establish some general results that are used multiple times in the upcoming proofs.

1.1.1 Equivalence Relations

In general, we use the symbol $\equiv$ to denote equivalence relations, mostly between states of an automata. In general, we have automata $\mathcal{A}$ and $\mathcal{B}$ with states p and q from there respective state spaces. Our relations are then defined on $(\mathcal{A}, p) \equiv (\mathcal{B}, q)$.

Definition 1.1.1. Assuming that $\mathcal{A}$ is a fixed automaton that is obvious in context and p and q are both states in $\mathcal{A}$, we shorten $(\mathcal{A}, p) \equiv (\mathcal{A}, q)$ to $p \equiv q$.

Furthermore, we write $\mathcal{A} \equiv \mathcal{B}$ if for every p in $\mathcal{A}$ there is a q in $\mathcal{B}$ such that $(\mathcal{A}, p) \equiv (\mathcal{B}, q)$; and the same holds with $\mathcal{A}$ and $\mathcal{B}$ exchanged.

Definition 1.1.2. Let $\mathcal{A} = (Q_1, \Sigma, \delta_1)$ and $\mathcal{B} = (Q_2, \Sigma, \delta_2)$ be deterministic transition structures and let $\sim \subseteq (\{\mathcal{A}\} \times Q_1) \times (\{\mathcal{B}\} \times Q_2)$ be an equivalence relation. We call R a *congruence relation* if for all $(\mathcal{A}, p) \sim (\mathcal{B}, q)$ and all $a \in \Sigma$, also $(\mathcal{A}, \delta_1(p, a)) \sim (\mathcal{B}, \delta_2(q, a))$.

The following is a comprehensive list of all relevant equivalence relations that we use.

- Language equivalence, $\equiv_L$. Defined below.
- Moore equivalence, $\equiv_M$. Defined below.
- Priority almost equivalence, $\equiv_\dagger$. Defined below.
- Delayed simulation equivalence, $\equiv_{\mathrm{de}}$. Defined in section 1.3.
- Delayed simulation equivalence with resets, $\equiv_{\mathrm{deR}}$. Defined in section 1.3.
- Iterated Moore equivalence, $\equiv_{\mathrm{IM}}$. Defined in section 1.4.
- Path refinement equivalence, $\equiv_{\mathrm{PR}}$. Defined in section 1.5.

- Threshold Moore equivalence, $\equiv_{\mathrm{TM}}^{\sim}$. Defined in section 1.6.
- Labeled SCC filter equivalence, $\equiv_{\mathrm{LSF}}^{k,\sim}$. Defined in section 1.7.

Immediately we define the three first of these relations and show that they are computable.

Language Equivalence

Definition 1.1.3. Let $\mathcal{A}$ and $\mathcal{B}$ be ω-automata. We define *language equivalence* as $(\mathcal{A}, p) \equiv_L (\mathcal{B}, q)$ if and only if for all words $\alpha \in \Sigma^\omega$, $\mathcal{A}$ accepts α from p iff $\mathcal{B}$ accepts α from q.

Lemma 1.1.1. *$\equiv_L$ is a congruence relation.*

Proof. It is obvious that $\equiv_L$ is an equivalence relation. For two states $(\mathcal{A}, p) \equiv_L (\mathcal{B}, q)$ and some successors $p' = \delta_1(p, a)$ and $q' = \delta_2(q, a)$, it must be true that $(\mathcal{A}, p') \equiv_L (\mathcal{B}, q')$. Otherwise there is a word $\alpha \in \Sigma^\omega$ that is accepted from p' and rejected from q' (or vice-versa). Then $a \cdot \alpha$ is rejected from p and accepted from q and thus $p \not\equiv_L q$. □

Lemma 1.1.2. *Language equivalence of a given DPA can be computed in $\mathcal{O}(|Q|^2 \cdot |c(Q)|^2)$.*

Proof. The algorithm is based partially on [7].

Let $\mathcal{A} = (Q, \Sigma, \delta, c)$ be the DPA that we want to compute $\equiv_L$ on. We construct a labeled deterministic transition structure $\mathcal{B} = (Q \times Q, \Sigma, \delta', d)$ with $\delta'((p_1, p_2), a) = (\delta(p_1, a), \delta(p_2, a))$ and $d((p_1, p_2)) = (c(p_1), c(p_2)) \in \mathbb{N}^2$. Then, for every $i, j \in c(Q)$, let $\mathcal{B}_{i,j} = \mathcal{B} \restriction_{Q_{i,j}}$ with $Q_{i,j} = \{(p_1, p_2) \in Q \times Q \mid c(p_1) \geq i, c(p_2) \geq j\}$, i.e. remove all states which have first priority less than i or second priority less than j.

For each i and j, let $S_{i,j} \subseteq 2^{Q \times Q}$ be the set of all SCCs in $\mathcal{B}_{i,j}$ and let $S = \bigcup_{i,j} S_{i,j}$. From this set S, remove all SCCs $s \subseteq Q \times Q$ in which the parity of the smallest priority in the first component differs from the parity of the smallest priority in the second component. The "filtered" set we call S'. For any two states $p, q \in Q$, $p \not\equiv_L q$ iff there is a pair $(p', q') \in \bigcup S'$ that is reachable from (p, q) in $\mathcal{B}$.

We omit the correctness proof of the algorithm here. Regarding the runtime, observe that $\mathcal{B}$ has size $\mathcal{O}(|Q|^2)$ and we create $\mathcal{O}(|c(Q)|^2)$ copies of it. All other steps like computing the SCCs can then be done in linear time in the size of the automata, which brings the total to $\mathcal{O}(|Q|^2 \cdot |c(Q)|^2)$. □

Priority Almost Equivalence

Definition 1.1.4. Let $\mathcal{A} = (Q_1, \Sigma, \delta_1, c_1)$ and $\mathcal{B} = (Q_2, \Sigma, \delta_2, c_2)$ be DPAs. We define *priority almost equivalence* as $(\mathcal{A}, p) \equiv_\dagger (\mathcal{B}, q)$ if and only if for all words $\alpha \in \Sigma^\omega$, $c_1^*(p, \alpha)$ and $c_2^*(q, \alpha)$ differ at only finitely many positions.

Lemma 1.1.3. *Priority almost equivalence is a congruence relation.*

Proof. It is obvious that $\equiv_\dagger$ is an equivalence relation. For two states $(\mathcal{A}, p) \equiv_\dagger (\mathcal{B}, q)$ and some successors $p' = \delta(p, a)$ and $q' = \delta(q, a)$, it must be true that $(\mathcal{A}, p') \equiv_\dagger (\mathcal{B}, q')$. Otherwise there is a word $\alpha \in \Sigma^\omega$ such that $c_1^*(p', \alpha)$ and $c_2^*(q', \alpha)$ differ at infinitely many positions. Then $c_1^*(p, a\alpha)$ and $c_2^*(q, a\alpha)$ also differ at infinitely many positions and thus $(\mathcal{A}, p) \not\equiv_\dagger (\mathcal{B}, q)$. □

The following definition is used as an intermediate step on the way to computing $\equiv_\dagger$.

Definition 1.1.5. Let $\mathcal{A} = (Q_1, \Sigma, \delta_1, c_1)$ and $\mathcal{B} = (Q_2, \Sigma, \delta_2, c_2)$ be DPAs. We define the deterministic Büchi automaton $\mathcal{A} \intercal \mathcal{B} = (Q_1 \times Q_2, \Sigma, \delta_\intercal, F_\intercal)$ with $\delta_\intercal((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$. The transition structure is a common product automaton.

The final states are $F_\intercal = \{(p, q) \in Q_1 \times Q_2 \mid c_1(p) \neq c_2(q)\}$, i.e. every pair of states at which the priorities differ.

Lemma 1.1.4. *$\mathcal{A} \intercal \mathcal{B}$ can be computed in time $\mathcal{O}(|Q_1| \cdot |Q_2|)$.*

Proof. The definition already provides a rather straightforward description of how to compute $\mathcal{A}\intercal\mathcal{B}$. Each state only requires constant time (assuming that δ and c can be evaluated in such) and has $|\mathcal{A}| \cdot |\mathcal{B}|$ many states. □

Lemma 1.1.5. *Let $\mathcal{A} = (Q_1, \Sigma, \delta_1, c_1)$ and $\mathcal{B} = (Q_2, \Sigma, \delta_2, c_2)$ be DPAs. $(\mathcal{A}, p) \equiv_\dagger (\mathcal{B}, q)$ iff $L(\mathcal{A} \intercal \mathcal{B}, (p, q)) = \emptyset$.*

Proof. For the first direction of implication, let $L(\mathcal{A} \intercal \mathcal{B}, (p_0, q_0)) \neq \emptyset$, so there is a word α accepted by that automaton. Let $(p, q)(p_1, q_1)(p_2, q_2) \cdots$ be the accepting run on α. Then $pp_1 \cdots$ and $qq_1 \cdots$ are the runs of $\mathcal{A}$ and $\mathcal{B}$ on α respectively. Whenever $(p_i, q_i) \in F_\intercal$, p_i and q_i have different priorities. As the run of the product automaton vists infinitely many accepting states, α is a witness for p and q being not priority almost-equivalent.

For the second direction, let p and q be not priority almost-equivalent, so there is a witness α at which infinitely many positions differ in priority. Analogously to the first direction, this means that the run of $\mathcal{A} \intercal \mathcal{B}$ on the same word is accepting and therefore the language is not empty. □

Corollary 1.1.6. *Priority almost equivalence of a given DPA can be computed in quadratic time.*

Proof. By Lemma 1.1.4, we can compute $\mathcal{A} \intercal \mathcal{A}$ in quadratic time. The emptiness problem for deterministic Büchi automata is solvable in linear time by checking reachability of loops that contain a state in F. □

Moore Equivalence

Definition 1.1.6. Let $\mathcal{A} = (Q_1, \Sigma, \delta_1, c_1)$ and $\mathcal{B} = (Q_2, \Sigma, \delta_2, c_2)$ be DPAs. We define *Moore equivalence* as $(\mathcal{A}, p) \equiv_M (\mathcal{B}, q)$ if and only if for all words $w \in \Sigma^*$, $c_1(\delta^*(p, w)) = c_2(\delta^*(q, w))$.

Lemma 1.1.7. *$\equiv_M$ is a congruence relation.*

Proof. It is obvious that $\equiv_M$ is an equivalence relation. For two states $(\mathcal{A}, p) \equiv_M (\mathcal{B}, q)$ and some successors $p' = \delta(p, a)$ and $q' = \delta(q, a)$, it must be true that $(\mathcal{A}, p') \equiv_M (\mathcal{B}, q')$. Otherwise there is a word $w \in \Sigma^*$ such that $c_1(\delta_1^*(p', w)) \neq c_2(\delta_2^*(q', w))$. Then $c_1(\delta_1^*(p, aw)) \neq c_2(\delta_2^*(q, aw))$ and thus $(\mathcal{A}, p) \not\equiv_M (\mathcal{B}, q)$. □

Lemma 1.1.8. *Moore equivalence of a given DPA can be computed in log-linear time.*

Proof. We refer to [8]. The given algorithm can be adapted to Moore automata without changing the complexity. □

Theorem 1.1.9. $\equiv_M \subseteq \equiv_\dagger \subseteq \equiv_L$

Proof. Let $\mathcal{A} = (Q_1, \Sigma, \delta_1, c_1)$ and $\mathcal{B} = (Q_2, \Sigma, \delta_2, c_2)$ be DPAs with states $q_1 \in Q_1$ and $q_2 \in Q_2$.

At first, let $(\mathcal{A}, q_1) \equiv_M (\mathcal{B}, q_2)$ and assume towards a contradiction that $(\mathcal{A}, q_1) \not\equiv_\dagger (\mathcal{B}, q_2)$, so there is a word $\alpha \in \Sigma^\omega$ such that $c_1^*(q_1, \alpha)$ and $c_2^*(q_2, \alpha)$ differ at infinitely many positions. In particular, there is some $w \sqsubseteq \alpha$ such that $c_1(\delta_1^*(q_1, w)) \neq c_2(\delta_2^*(q_2, w))$. This would be a contradiction to $(\mathcal{A}, q_1) \equiv_M (\mathcal{B}, q_2)$.

Now assume that $(\mathcal{A}, q_1) \equiv_\dagger (\mathcal{B}, q_2)$ and let $\alpha \in \Sigma^\omega$. Let ρ_1 and ρ_2 be the runs of $\mathcal{A}$ and $\mathcal{B}$ on α starting in q_1 and q_2. Because $(\mathcal{A}, q_1) \equiv_\dagger (\mathcal{B}, q_2)$ is true, there is a position n such that $c_1(\rho_1[n, \omega]) = c_2(\rho_2[n, \omega])$ and therefore $\mathrm{Inf}(c_1(\rho_1[n, \omega])) = \mathrm{Inf}(c_2(\rho_2[n, \omega]))$, which means that the runs have the same acceptance. Thus, $(\mathcal{A}, q_1) \equiv_L (\mathcal{B}, q_2)$. □

1.1.2 Representative Merge

Definition 1.1.7. Let $\mathcal{A} = (Q, \Sigma, \delta, c)$ be a DPA and let $\emptyset \neq C \subseteq M \subseteq Q$. Let $\mathcal{A}' = (Q', \Sigma, \delta', c')$ be another DPA. We call $\mathcal{A}'$ a *representative merge of $\mathcal{A}$ w.r.t. M by candidates C* if it satisfies the following:

- There is a state $r_M \in C$ such that $Q' = (Q \setminus M) \cup \{r_M\}$.
- $c' = c \restriction_{Q'}$.
- Let $p \in Q'$ and $\delta(p, a) = q$. If $q \in M$, then $\delta'(p, a) = r_M$. Otherwise, $\delta'(p, a) = q$.

We call r_M the *representative* of M in the merge. We might omit C and implicitly assume $C = M$.

Definition 1.1.8. Let $\mathcal{A} = (Q, \Sigma, \delta, c)$ be a DPA and let $\mu : D \to (2^Q \setminus \emptyset)$ be a function for some $D \subseteq 2^Q$. We call μ a *merger function* if

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

A DPA $\mathcal{A}'$ is a representative merge of $\mathcal{A}$ w.r.t. μ if there is an enumeration $X_1, \ldots, X_{|D|}$ of D and a sequence of automata $\mathcal{A}_0, \ldots, \mathcal{A}_{|D|}$ such that $\mathcal{A}_0 = \mathcal{A}$, $\mathcal{A}_{|D|} = \mathcal{A}'$ and every $\mathcal{A}_{i+1}$ is a representative merge of $\mathcal{A}_i$ w.r.t. X_{i+1} by candidates $\mu(X_{i+1})$.

The following Lemma formally proofs that this definition actually makes sense, as building representative merges is commutative if the merge sets are disjoint.

Lemma 1.1.10. *Let $\mathcal{A} = (Q, \Sigma, \delta, c)$ be a DPA and let $M_1, M_2 \subseteq Q$. Let $\mathcal{A}_1$ be a representative merge of $\mathcal{A}$ w.r.t. M_1 by some candidates C_1. Let $\mathcal{A}_{12}$ be a representative merge of $\mathcal{A}_1$ w.r.t. M_2 by some candidates C_2. If M_1 and M_2 are disjoint, then there is a representative merge $\mathcal{A}_2$ of $\mathcal{A}$ w.r.t. M_2 by candidates C_2 such that $\mathcal{A}_{12}$ is a representative merge of $\mathcal{A}_2$ w.r.t M_1 by candidates C_1.*

Proof. By choosing the same representative r_{M_1} and r_{M_2} in the merges, this is a simple application of the definition. □

The following Lemma, while simple to prove, is interesting and will find use in multiple proofs of correctness later on.

Lemma 1.1.11. *Let $\mathcal{A}$ be a DPA. Let $\sim$ be a congruence relation on Q and let $M \subseteq Q$ such that for all $x, y \in M$, $x \sim y$. Let $\mathcal{A}'$ be a representative merge of $\mathcal{A}$ w.r.t. M by candidates C. Let ρ and ρ' be runs of $\mathcal{A}$ and $\mathcal{A}'$ on some α. Then for all i, $(\mathcal{A}, \rho(i)) \sim (\mathcal{A}, \rho'(i))$.*

Proof. We use a proof by induction. For $i = 0$, we have $\rho(0) = q_0$ for some $q_0 \in Q$ and $\rho'(0) = r_{[q_0]_M}$. By choice of the representative, $q_0 \in M$ and $r_{[q_0]_M} \in M$ and thus $q_0 \sim r_{[q_0]_M}$.

Now consider some $i + 1 > 0$. Then $\rho'(i+1) = r_{[q]_M}$ for $q = \delta(\rho'(i), \alpha(i))$. By induction we know that $\rho(i) \sim \rho'(i)$ and thus $\delta(\rho(i), \alpha(i)) = \rho(i+1) \sim q$. Further, we know $q \sim r_{[q]_M}$ by the same argument as before. Together this lets us conclude in $\rho(i+1) \sim q \sim \rho'(i+1)$. $\square$

The following is a comprehensive list of all relevant merger functions that we use.

- Quotient merger, $\mu^{\sim}_{\div}$. Defined below.
- Moore merger, μ_M. Defined below.
- Skip merger, $\mu^{\sim}_{\text{skip}}$. Defined in section 1.2.
- Delayed simulation merger, μ_{de}. Defined in section 1.3.
- Path refinement metger
- Treshold Moore merger, $\mu^{\sim}_{\text{TM}}$. Defined in section 1.6.
- Labeled SCC Filter merger, $\mu^{k,\sim}_{\text{LSF}}$. Defined in section 1.7.

Quotient merger

Definition 1.1.9. Let $\sim$ be a congruence relation. We define the *quotient merger* $\mu^{\sim}_{\div} : \mathfrak{C}(\sim) \to 2^Q, \kappa \mapsto \kappa$.

Lemma 1.1.12. *Let $\mathcal{A} = (Q, \Sigma, \delta, c)$ be a DPA and let $\sim$ be a congruence relation such that $p \sim q$ implies $c(p) = c(q)$. Then $\mathcal{A}$ is Moore equivalent to every representative merge w.r.t. $\mu^{\sim}_{\div}$.*

Proof. Let $\mathcal{A}' = (Q', \Sigma, \delta', c')$ be a representative merge. For every $q \in Q$, we prove that $(\mathcal{A}, q) \equiv_M (\mathcal{A}', r_{[q]_\sim})$. Since all states in $\mathcal{A}'$ are representatives of that form and every representative exists in $\mathcal{A}$ as well, this suffices to prove Moore equivalence.

Let $\alpha \in \Sigma^\omega$ be a word and let ρ and ρ' be the runs of $\mathcal{A}$ and $\mathcal{A}'$ on α starting in q and $r_{[q]_\sim}$. For every $i \in \mathbb{N}$, we have $\rho'(i) = [\rho(i)]_\sim$ and thus $c'(\rho'(i)) = c(\rho(i))$. Therefore, ρ is accepting iff ρ' is accepting. $\square$

Definition 1.1.10. We define the special *Moore merger* $\mu_M = \mu^{\equiv_M}_{\div}$.

1.1.3 Reachability

Definition 1.1.11. Let $\mathcal{S} = (Q, \Sigma, \delta)$ be a deterministic transition structure. We define the *reachability order* $\preceq^{\mathcal{S}}_{\text{reach}}$ as $p \preceq^{\mathcal{S}}_{\text{reach}} q$ if and only if q is reachable from p.

We want to note here that we always assume for all automata to only have one connected component, i.e. for all states p and q, there is a state r such that p and q are both reachable from r. In practice, most automata have an predefined initial state and a simple depth first search can be used to eliminate all unreachable states.

Lemma 1.1.13. *$\preceq^{\mathcal{S}}_{\text{reach}}$ is a preorder.*

Definition 1.1.12. Let $\mathcal{S} = (Q, \Sigma, \delta)$ be a deterministic transition structure. We call a relation $\preceq$ a *total extension of reachability* if it is a minimal superset of $\preceq^{\mathcal{S}}_{\text{reach}}$ that is also a total preorder.

For $p \preceq q$ and $q \preceq p$, we write $p \simeq q$.

Lemma 1.1.14. *For a given deterministic transition structure $\mathcal{S}$, a total extension of reachability is computable in $\mathcal{O}(|\mathcal{S}|)$.*

Proof. Using e.g. Kosaraju's algorithm [16], the SCCs of $\mathcal{A}$ can be computed in linear time. We can now build a DAG from $\mathcal{A}$ by merging all states in an SCC into a single state; iterate over all transitions (p, a, q) and add an a-transition from the merged representative of p to that of q. Assuming efficient data structures for the computed SCCs, this DAG can be computed in $O(|\mathcal{A}|)$ time.

To finish the computation of $\preceq$, we look for a topological order on that DAG. This is a total preorder on the SCCs that is compatible with reachability. All that is left to be done is to extend that order to all states. □

1.1.4 Changing Priorities

As we mentioned earlier, state reduction of DPAs is difficult and minimization is an NP-hard problem. Priorities of states on the other hand are generally easier to modify. A few of these possibilities are considered in this section.

Lemma 1.1.15. *Let $\mathcal{A} = (Q, \Sigma, \delta, c)$ be a DPA and let $\{s\} \subseteq Q$ be a trivial SCC in $\mathcal{A}$. Let $\mathcal{A}' = (Q, \Sigma, \delta, c')$ be a copy of $\mathcal{A}$ with the only exception that $c'(s) = k$ for some arbitrary k. Then $\mathcal{A} \equiv_L \mathcal{A}'$.*

Proof. Let $q \in Q$ be any state. We show that $L(\mathcal{A}, q) = L(\mathcal{A}', q)$. Let ρ and ρ' be the runs of $\mathcal{A}$ and $\mathcal{A}'$ starting in q on some $\alpha \in \Sigma^\omega$. As s lies in a trivial SCC, the runs visit that state at most once. Therefore, $\text{Inf}c(\rho) = \text{Inf}c'(\rho')$ and the runs have the same acceptance. □

An interesting property that parity automata can have is being *normalized*. Even more important is that a normalized version of a DPA can be computed rather easily.

Definition 1.1.13. Let $\mathcal{A} = (Q, \Sigma, \delta, c)$ be a DPA. We call $\mathcal{A}$ *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.

Algorithm 1 shows how an equivalent normalized priority function can be computed in $\mathcal{O}(|Q| \cdot |c(Q)|)$. The algorithm is a slight adaption of that presented in [3], which is why we will not go into further details here and just refer to the original source.

Algorithm 1 Normalizing the priority function of a DPA.

```
1: function NORMALIZE($\mathcal{A}$)
2:     $c' : Q \to \mathbb{N}, q \mapsto c(q)$
3:     M($\mathcal{A}$, $c'$)
4:     return $c'$
5: end function

6: function M($\mathcal{A} \upharpoonright_P$, $c'$)
7:     if $P = \emptyset$ then
8:         return 0
9:     end if
10:    $min \leftarrow 0$
11:    for SCC $S$ in $\mathcal{A} \upharpoonright_P$ do
12:        $m := \min c(S) \mod 2$
13:        $X := c^{-1}(m)$
14:        for $q \in X$ do
15:            $c'(q) \leftarrow m$
16:        end for
17:        $S' := S \setminus X$
18:        $m' \leftarrow$M($\mathcal{A} \upharpoonright_{S'}$, $c'$)
19:        if $m'$ even then
20:            if $m$ even then
21:                $\delta := m$
22:            else
23:                $\delta := m - 2$
24:            end if
25:        else
26:            $\delta := m - 1$
27:        end if
28:        for $q \in S'$ do
29:            $c'(q) \leftarrow c'(q) - \delta$
30:        end for
31:        $min \leftarrow \min\{min, m\}$
32:    end for
33:    return $min$
34: end function
```

1.2 Congruence Path Refinement

Definition 1.2.1. Let $\mathcal{A} = (Q, \Sigma, \delta, c)$ be a DPA and let $R \subseteq Q \times Q$ be a congruence relation on the state space. Let $\lambda \subseteq Q$ be an equivalence class of R. We define $L_{\lambda\hookleftarrow}$ as the set of non-empty words w such that for any $u \sqsubseteq w$, $(\delta(p, u), q) \in R$ iff $u \in \{\varepsilon, w\}$. In other words, the set contains all minimal words by which the automaton moves from λ to λ again.

Let $f_{\mathrm{PR}} : 2^{\lambda\times\lambda} \to 2^{\lambda\times\lambda}$ be a function such that $(p, q) \in f(X)$ iff for all $w \in L_{\lambda\hookleftarrow}$, $(\delta^*(p, w), \delta^*(q, w)) \in X$. Then let $X_0 \subseteq \lambda \times \lambda$ such that $(p, q) \in X_0$ iff for all $w \in L_{\lambda\hookleftarrow}$, $\min\{c(\delta^*(p, u)) \mid u \sqsubseteq w\} = \min\{c(\delta^*(q, u)) \mid u \sqsubseteq w\}$, i.e. the minimal priority when moving from p or q to λ again is the same.

Using both, we set $X_{i+1} = f_{\mathrm{PR}}(X_i)$. f_{PR} is monotone w.r.t. $\subseteq$, so there is an $X_n = X_{n+1}$ by Kleene's fixed point theorem. We define the *path refinement of* λ, called $\equiv^{\lambda}_{\mathrm{PR}}$, as

- For $p \in Q \setminus \lambda$, $p \equiv^{\lambda}_{\mathrm{PR}} q$ iff $p = q$.
- For $p, q \in \lambda$, $p \equiv^{\lambda}_{\mathrm{PR}} q$ iff $(p, q) \in X_n$.

Definition 1.2.2. Let $\mathcal{A}$ be a DPA. We define the *path refinement merger*

Theorem 1.2.1. *Let $\mathcal{A} = (Q, \Sigma, q_0, \delta, c)$ be a DPA and let $R \subseteq Q \times Q$ be a congruence relation that implies language equivalence. Let $\mathcal{A}'$ be a representative merge of $\mathcal{A}$ w.r.t. $\equiv^{\lambda}_{\mathrm{PR}}$ for some equivalence class λ of R such that each representative r_κ is chosen to have minimal priority. Then $L(\mathcal{A}) = L(\mathcal{A}')$.*

Proof. Let $\alpha \in \Sigma^\omega$ be a word with runs $\rho \in Q^\omega$ and $\rho' \in (Q')^\omega$ of $\mathcal{A}$ and $\mathcal{A}'$ respectively. Let $k_0, \dots \in \mathbb{N}$ be exactly those positions (in order) at which ρ reaches λ, and analogously $k'_0, \dots$ for ρ'.

Claim 1: For every i, $k_i = k'_i$ and $\rho(k_i) \equiv^{\lambda}_{\mathrm{PR}} \rho'(k_i)$.
For all $j < k_0$, we know that $\rho(j) = \rho'(j)$, as no redirected edge is taken. Thus, $\rho'(k_0) = r_{[\rho(k_0)]_{\equiv^{\lambda}_{\mathrm{PR}}}} \equiv^{\lambda}_{\mathrm{PR}} = \rho(k_0)$.

Now assume that the claim holds for all $i \leq n$. By definition, $w = \alpha[k_n, k_{n+1}] \in L_{\lambda\hookleftarrow}$ and therefore $\rho(k_{n+1}) = \delta^*(\rho(k_n), w) \equiv^{\lambda}_{\mathrm{PR}} \delta^*(\rho'(k_n), w) = \rho'(k_{n+1})$.

Claim 2: If λ only occurs finitely often in ρ and ρ', then ρ is accepting iff ρ' is accepting.

Let $k_n \in \mathbb{N}$ be the last position at which $\rho(k_n)$ and $\rho'(k_n)$ are in λ. From this point on, $\rho'[k_n, \omega]$ is also a valid run of $\mathcal{A}$ on $\alpha[k_n, \omega]$. $\rho(k_n), \rho'(k_n) \in \lambda$, so $(\rho(k_n), \rho'(k_n)) \in R$. As R implies language equivalence, reading $\alpha[k_n, \omega]$ from either state in $\mathcal{A}$ leads to the same acceptance status. This also means that $\rho'(k_n)$ has the same acceptance status as $\rho(k_n)$.

Claim 3: If λ occurs infinitely often in ρ and ρ', then ρ is accepting iff ρ' is accepting.

We show that for all i, $\min \mathrm{Occ}(c(\rho[k_i, k_{i+1} + 1]))$ and $\min \mathrm{Occ}(c'(\rho'[k_i, k_{i+1} + 1]))$ are the same. The claim then follows immediately.

Directly observe that $c'(\rho'[k_i, k_{i+1} + 1]) = c(\rho'[k_i, k_{i+1} + 1])$ and that $\min \mathrm{Occ}(c(\rho[k_i, k_{i+1} + 1])) = \min \mathrm{Occ}(c(\rho'[k_i, k_{i+1}] \cdot \delta(\rho'(k_{i+1} - 1), \alpha(k_{i+1}))))$ because $\rho(k_i) \equiv^{\lambda}_{\mathrm{PR}} \rho'(k_i)$.

Now $\mathrm{Occ}(c(\rho[k_i, k_{i+1}+1])) = \mathrm{Occ}(c(\rho[k_i, k_{i+1}])) \cup \{c(\rho(k_{i+1}))\}$ and $\mathrm{Occ}(c(\rho'[k_i, k_{i+1}] \cdot \delta(\rho'(k_{i+1} - 1), \alpha(k_{i+1})))) = \mathrm{Occ}(c(\rho[k_i, k_{i+1}])) \cup \{c(\delta(\rho'(k_{i+1} - 1), \alpha(k_{i+1})))\}$.

The rest of the claim follows because $c(\rho'(k_i)) = c(\rho'(k_{i+1})) \leq \delta(\rho'(k_{i+1}, \alpha(k_{i+1}))$. $\square$

1.2.1 Algorithmic Definition

The definition of path refinement that we introduced is useful for the proofs of correctness. It however does not provide one with a way to actually compute the relation. That is why we now provide an alternative definition that yields the same results but is more algorithmic in nature.

Definition 1.2.3. Let $\mathcal{A} = (Q, \Sigma, \delta, c)$ be a DPA and let $R \subseteq Q \times Q$ be a congruence relation. For each equivalence class λ of R, we define the *path refinement automaton* $\mathcal{G}_{\mathrm{PR}}^{R,\lambda}(p, q) = (Q_{\mathrm{PR}}, \Sigma, \delta_{\mathrm{PR}}^{\lambda}, F_{\mathrm{PR}})$, which is a DFA.

- $Q_{\mathrm{PR}} = (Q \times Q \times c(Q) \times \{<, >, =\}) \cup \{\bot\}$
- The initial state we use is $q_0^{\mathrm{PR}}(p, q) = (p, q, \eta_k(c(p), c(q), \checkmark), \eta_x(c(p), c(q), \checkmark, =))$
- $$\delta_{\mathrm{PR}}^{\lambda}((p, q, k, x), a) = \begin{cases} (p', q', \eta_k(c(p'), c(q'), k), \eta_x(c(p'), c(q'), k, x)) & \text{if } p' \notin \lambda \\ q_0^{\mathrm{PR}}(p', q') & \text{if } p' \in \lambda \text{ and } (\eta_x(c(p'), c(q'), k, x) = =) \\ \bot & \text{else} \end{cases}$$
  where $p' = \delta(p, a)$ and $q' = \delta(q, a)$.
  $\eta_k(k_p, k_q, k) = \min_{\leq_\checkmark}\{k_p, k_q, k\}$
  $$\eta_x(k_p, k_q, k, x) = \begin{cases} < & \text{if } (k_p <_\checkmark k_q \text{ and } k_p <_\checkmark k) \text{ or } (k < k_q \text{ and } (x = <)) \\ > & \text{if } (k_p >_\checkmark k_q \text{ and } k >_\checkmark k_q) \text{ or } (k_p > k \text{ and } (x = >)) \\ = & \text{else} \end{cases}$$
- $F_{\mathrm{PR}} = Q_{\mathrm{PR}} \setminus \{\bot\}$

Lemma 1.2.2. *Let $\mathcal{A}$ be a DPA with a congruence relation R. Let λ be an equivalence class of R, $p, q \in \lambda$, and $w \in L_{\lambda\to\lambda}$. For every $v \sqsubset w$ and $\oplus \in \{<, >, =\}$, the fourth component of $(\delta_{PR}^{\lambda})^*(q_0^{PR}(p, q), v)$ is $\oplus$ if and only if* $\min\{c(\delta^*(p, u)) \mid u \sqsubseteq v\} \oplus \min\{c(\delta^*(q, u)) \mid u \sqsubseteq v\}$.

The proof of this Lemma is a very formal analysis of every case in the relations between the different priorities that occur and making sure that the definition of η_x covers these correctly. No great insight is gained, which is why we omit the proof at this point.

Theorem 1.2.3. *Let $\mathcal{A}$ be a DPA with a congruence relation R. Let λ be an equivalence class of R and $p, q \in \lambda$. Then $p \equiv_{\mathrm{PR}}^{R} q$ iff $L(\mathcal{G}_{\mathrm{PR}}^{R,\lambda}(p, q), q_0^{PR}(p, q)) = \Sigma^*$.*

Proof. **If** Let $p \not\equiv_{\mathrm{PR}}^{R} q$. We use the inductive definition of $R_\kappa \subseteq \equiv_{\mathrm{PR}}^{R}$ using f and the sets X_i here. Let m be the smallest index at which $(p, q) \notin X_m$. Let $\rho = (p_i, q_i, k_i, x_i)_{0 \leq i \leq |w|}$ be the run of $\mathcal{G}_{\mathrm{PR}}^{R,\lambda}(p, q)$ on w. We prove that $\rho(|w|) = \bot$ and therefore ρ is not accepting by induction on m.

If $m = 0$, then $(p, q) \notin Y_\lambda$, meaning that there is a word w such that $\min\{c(\delta^*(p, u)) \mid u \sqsubset w\} \neq \min\{c(\delta^*(q, u)) \mid u \sqsubset w\}$. Without loss of generality, assume $\min\{c(\delta^*(p, u)) \mid u \sqsubset w\} < \min\{c(\delta^*(q, u)) \mid u \sqsubset w\}$. By Lemma 1.5.2, $x_{|w|-1} = <$. Furthermore, $\delta(p_{|w|-1}, w_{|w|-1}) \in \lambda$, as $w \in L_{\lambda\to\lambda}$. Thus, $\rho(|w|) = \bot$ and the run is rejecting.

Now consider $m + 1 > 1$. Since $(p, q) \in X_m \setminus f(X_m)$, there must be a word $w \in L_{\lambda\to\lambda}$ such that $(p', q') \notin X_m$, where $p' = \delta^*(p, w)$ and $q' = \delta^*(q, w)$. As $R_\kappa \subseteq X_m$, $(p', q') \notin R_\kappa$ and therefore $p' \not\equiv_{\mathrm{PR}}^{R} q'$. By induction, $w \notin L(\mathcal{G}_{\mathrm{PR}}^{R,\lambda}(p', q'), q_0^{\mathrm{PR}}(p', q'))$; since that run is a suffix of ρ, ρ itself is also a rejecting run.

Only If Let $L(\mathcal{G}^{R,\lambda}_{\text{PR}}(p,q), q_0^{\text{PR}}(p,q)) \neq \Sigma^*$. Since ε is always accepted, there is a word $w \in \Sigma^+ \setminus L(\mathcal{G}^{R,\lambda}_{\text{PR}}(p,q), q_0^{\text{PR}}(p,q))$, meaning that $\delta^*_{\text{PR}}(q_0^{\text{PR}}(p,q), w) = \perp$. Split w into sub-words $w = u_1 \cdots u_m$ such that $u_1, \ldots, u_m \in L_{\lambda \to \lambda}$. Note that this partition is unique. We show $p \not\equiv^R_{\text{PR}} q$ by induction on m. Let $\rho = (p_i, q_i, k_i, x_i)_{0 \le i < |w|}$ be the run of $\mathcal{G}^{R,\lambda}_{\text{PR}}(p,q)$ on w starting in $q_0^{\text{PR}}(p,q)$.

If $m = 1$, then $w \in L_{\lambda \to \lambda}$. Since $\rho(|w|) = \perp$, it must be true that $x_{|w|-1} \neq =$. Without loss of generality, assume $x_{|w|-1} = <$. By Lemma 1.5.2, $\min\{c(\delta^*(p,u)) \mid u \sqsubseteq w\} < \min\{c(\delta^*(q,u)) \mid u \sqsubseteq w\}$. Therefore, $p \not\equiv^R_{\text{PR}} q$.

Now consider $m + 1 > 1$. Let $p' = \delta^*(p, u_1)$ and $q' = \delta^*(q, u_1)$. By induction on the word $u_2 \cdots u_m$, $p' \not\equiv^R_{\text{PR}} q'$. Since $u_1 \in L_{\lambda \to \lambda}$, that also means $p \not\equiv^R_{\text{PR}} q$. □

The differences between different $\mathcal{G}^{R,\lambda}_{\text{PR}}$ for different λ are minor and the question whether the accepted language is universal boils down to a simple question of reachability. Thus, $\equiv^R_{\text{PR}}$ can be computed in $\mathcal{O}(|\mathcal{G}^{R,\lambda}_{\text{PR}}|)$ which is $\mathcal{O}(|Q|^2 \cdot |c(Q)|)$.

1.2.2 Alternative Algorithmic Definition

The computation presented in the previous section was a straight-forward description of $\equiv^\lambda_{\text{PR}}$ in an algorithmic way. We can reduce the complexity of that computation by taking a more indirect route, as we will see now.

Definition 1.2.4. Let $\mathcal{A} = (Q, \Sigma, \delta, c)$ be a DPA. Let R be a congruence relation on Q and let $\lambda \subseteq Q$ be an equivalence class of R. We define a deterministic transition structure $\mathcal{A}^\lambda_{\text{visit}} = (Q^\lambda_{\text{visit}}, \Sigma, \delta^\lambda_{\text{visit}})$ as follows:

- $Q^\lambda_{\text{visit}} = ((Q \setminus \lambda) \times c(Q) \times \{\perp\}) \cup (\lambda \times c(Q) \times c(Q))$
  These states "simulate" $\mathcal{A}$ and use the second component to track the minimal priority that was seen since a last visit to λ. The states in λ itself also have a third component that is used to distinguish their classes, as is explained below.
- $\delta^\lambda_{\text{visit}}((q,k,k'),a) = \begin{cases} (q', \min\{k, c(q')\}, \perp) & \text{if } q' \notin \lambda \\ (q', c(q'), \min\{k, c(q')\}) & \text{if } q' \in \lambda \end{cases}$, where $q' = \delta(q,a)$.

Definition 1.2.5. Consider $\mathcal{A}^\lambda_{\text{visit}}$ of a DPA $\mathcal{A}$ and a congruence relation R. We define an equivalence relation $V \subseteq Q^\lambda_{\text{visit}} \times Q^\lambda_{\text{visit}}$ as:

- For every $p, q \in Q \setminus \lambda$ and $l, k \in c(Q)$, $((p,l,\perp),(q,k,\perp)) \in V$.
- For every $p, q \in \lambda$ and $l, k \in c(Q)$, $((p,l,l'),(q,k,k')) \in V$ iff $l' = k'$.

The congruence refinement of V is then called V_M.
We abbreviate the state $(q, c(q), k)$ for any $q \in \lambda$ and $k \in c(Q)$ by ι^k_q.

Lemma 1.2.4. *For all $p, q \in \lambda$ and $l, k \in c(Q)$: $(\iota^l_p, \iota^k_q) \in V$ iff $l = k$.*

Proof. Follows directly from the definition. □

Lemma 1.2.5. *Let $q \in \lambda$, $k \in c(Q)$, $w \in L_{\lambda \hookleftarrow}$, and $\varepsilon \sqsubset v \sqsubset w$. Then $(\delta^\lambda_{\text{visit}})^*(\iota^k_q, v) = (\delta^*(q,v), x_v, \perp)$, where $x_v = \min\{c(\delta^*(q,u)) \mid u \sqsubseteq v\}$.*

Proof. We provide this proof by using induction on v. First, consider $v = a \in \Sigma$. Since $v \notin L_{\lambda\hookleftarrow}$, we know $\delta(q, a) \notin \lambda$ and thus $(\delta^\lambda_{\text{visit}})^*(\iota^k_q, v) = \delta^\lambda_{\text{visit}}(\iota^k_q, a) = (\delta(q, a), c(\min\{c(q), c(\delta(q, a))\}, \bot)$. This is exactly what we had to show, as $\min\{c(q), c(\delta(q, a))\} = x_a$.

For the induction step, let $v = v'a \in \Sigma^+ \cdot \Sigma$. Then $(\delta^\lambda_{\text{visit}})^*(\iota^k_q, v) = \delta^\lambda_{\text{visit}}((\delta^*(q, v'), x_{v'}, \bot), a)$ by induction. Again, $\delta^*(q, v) \notin \lambda$, so $(\delta^\lambda_{\text{visit}})^*(\iota^k_q, v) = (\delta^*(q, v), \min\{x_{v'}, c(\delta^*(q, v))\}, \bot)$. This is our goal, as $\min\{x_{v'}, c(\delta^*(q, v))\} = x_v$. □

Lemma 1.2.6. *Let $q \in \lambda$, $k \in c(Q)$, and $w \in L_{\lambda\hookleftarrow}$. Then $(\delta^\lambda_{\text{visit}})^*(\iota^k_q, w) = \iota^x_{q'}$, where $q' = \delta^*(q, w)$ and $x = \min\{c(\delta^*(q, u)) \mid u \sqsubseteq w\}$.*

Proof. For all $v \sqsubseteq w$, let $x_v = \min\{c(\delta^*(q, u)) \mid u \sqsubseteq v\}$ (i.e. $x = x_w$). Let $w = va \in \Sigma^* \cdot \Sigma$ (since words in $L_{\lambda\hookleftarrow}$ are non-empty). Then $(\delta^\lambda_{\text{visit}})^*(\iota^k_q, v) = (\delta^*(q, v), x_v, \bot)$ by Lemma 1.5.5 and $(\delta^\lambda_{\text{visit}})^*(\iota^k_q, w) = \delta^\lambda_{\text{visit}}((\delta^*(q, v), x_v, \bot), a)$.

Let $q' = \delta^*(q, w)$. Since $w \in L_{\lambda\hookleftarrow}$, $q' \in \lambda$ and definition tells us $\delta^\lambda_{\text{visit}}((\delta^*(q, v), x_v, \bot), a) = (q', c(q'), \min\{x_v, c(q')\})$. The fact that $\min\{x_v, c(q')\} = x_w$ finishes our proof. □

Lemma 1.2.7. *For every $q \in \lambda$, $l, k \in c(Q)$, and $w \in \Sigma^+$: $(\delta^\lambda_{\text{visit}})^*(\iota^k_q, w) = (\delta^\lambda_{\text{visit}})^*(\iota^l_q, w)$.*

Proof. If suffices to consider the case $w = a \in \Sigma$. If the statement is true for any one-symbol word, then it is for words of any length, as $\mathcal{A}^\lambda_{\text{visit}}$ is deterministic.

For $w \in \Sigma$, w is always in $L_{\lambda\hookleftarrow}$ or a prefix of a word in that set. Thus we can apply Lemma 1.5.5 and 1.5.6 to obtain our wanted result. □

Lemma 1.2.8. *For all $p, q \in \lambda$, and $l, k \in c(Q)$, $(\iota^k_p, \iota^k_q) \in V_M$ if and only if $(\iota^l_p, \iota^l_q) \in V_M$.*

Proof. As l and k are chosen symmetrically, it suffices for us to prove on direction of the bidirectional implication. Assume towards a contradiction that $(\iota^k_p, \iota^k_q) \in V_M$ but $(\iota^l_p, \iota^l_q) \notin V_M$, so there is a word $w \in \Sigma^*$ such that $((\delta^\lambda_{\text{visit}})^*(\iota^l_p, w), (\delta^\lambda_{\text{visit}})^*(\iota^l_q, w)) \notin V$.

It must be true that $w \neq \varepsilon$; otherwise, $(\iota^l_p, \iota^l_q) \notin V$, which would contradict Lemma 1.5.4.

By Lemma 1.5.7, we have $(\delta^\lambda_{\text{visit}})^*(\iota^l_p, w) = (\delta^\lambda_{\text{visit}})^*(\iota^k_p, w)$ and analogously for q. Therefore, $((\delta^\lambda_{\text{visit}})^*(\iota^k_p, w), (\delta^\lambda_{\text{visit}})^*(\iota^k_q, w)) \notin V$ and thus $(\iota^k_p, \iota^k_q) \notin V_M$, which contradicts our assumption. □

Lemma 1.2.9. *Let $q \in \lambda$ and $w \in \Sigma^+$ such that $\delta^*(q, w) \in \lambda$. Then there is a decomposition of w into words $v_1 \cdots v_m$ such that all $v_i \in L_{\lambda\hookleftarrow}$.*

Proof. Let $\rho \in Q^*$ be the run of $\mathcal{A}$ starting in q on w. Let $i_1 < \cdots < i_m$ be those positions at which $\rho(i_j) \in \lambda$. For every $1 \leq j < m$, we define $v_j = w[i_j, i_{j+1}]$. By choice of the i_j, all of those words are elements of $L_{\lambda\hookleftarrow}$.

Since $q \in \lambda$ and $\delta^*(q, w) \in \lambda$, we have $i_0 = 0$ and $i_m = |w| + 1$, so $w = v_1 \cdots v_m$. □

Theorem 1.2.10. *Let $\mathcal{A}$, R, and λ be as before. Let $\hat{c} = \max c(Q)$. Then for all $p, q \in \lambda$, we have $p \equiv^\lambda_{\text{PR}} q$ iff $(\iota^{\hat{c}}_p, \iota^{\hat{c}}_q) \in V_M$.*

Proof. We have $(\iota^{\hat{c}}_p, \iota^{\hat{c}}_q) \in V_M$ if and only if for all $w \in \Sigma^*$: $(\delta^*(\iota^{\hat{c}}_p, w), \delta^*(\iota^{\hat{c}}_q, w)) \in V$. Thus, we want to show that his property holds for all w iff $p \equiv^\lambda_{\text{PR}} q$.

If Assume there is a w such that $((\delta^{\lambda}_{\text{visit}})^*(\iota^{\hat{c}}_p, w), (\delta^{\lambda}_{\text{visit}})^*(\iota^{\hat{c}}_q, w)) \notin V$. Choose this w to have minimal length. By definition of V, both $(\delta^{\lambda}_{\text{visit}})^*(\iota^{\hat{c}}_p, w)$ and $(\delta^{\lambda}_{\text{visit}})^*(\iota^{\hat{c}}_q, w)$ must be in λ. By Lemma 1.5.9, there is a decomposition of w into words $v_1 \cdots v_m$ that are in $L_{\lambda\hookleftarrow}$. We perform a proof of induction on m.

If $m = 1$, then $w \in L_{\lambda\hookleftarrow}$. From Lemmas 1.5.4 and 1.5.6, we know that $\min\{c(\delta^*(p, u)) \mid u \sqsubseteq w\} \neq \min\{c(\delta^*(q, u)) \mid u \sqsubseteq w\}$ and therefore $p \not\equiv^{\lambda}_{\text{PR}} q$.

If $m + 1 > 1$, consider $(\delta^{\lambda}_{\text{visit}})^*(\iota^{\hat{c}}_q, v_1) = \iota^{x}_{q'}$ and $(\delta^{\lambda}_{\text{visit}})^*(\iota^{\hat{c}}_q, v_1) = \iota^{y}_{p'}$ as stated in Lemma 1.5.6 (with $p' = \delta^*(p, v_1)$ and q' analogously). w was chosen to have minimal length, so $(\iota^{x}_{q'}, \iota^{y}_{p'}) \in V$, which means that $x = y$.

As $(\iota^{\hat{c}}_p, \iota^{\hat{c}}_q) \notin V_M$, we also have $(\iota^{x}_{p'}, \iota^{y}_{q'}) \notin V_M$ and by Lemma 1.5.8, $(\iota^{\hat{c}}_{p'}, \iota^{\hat{c}}_{q'}) \notin V_M$ with the word $v_2 \cdots v_m$ being a witness. We can therefore argue with induction to deduce $p' = \delta^*(p, v_1) \not\equiv^{\lambda}_{\text{PR}} q' = \delta^*(q, v_1)$. As $v_1 \in L_{\lambda\hookleftarrow}$, the definition of path refinement tells us $p \not\equiv^{\lambda}_{\text{PR}} q$.

Only If Assume $p \not\equiv^{\lambda}_{\text{PR}} q$. Let f_{PR} and $(X_i)_i$ be the function and sets used in the construction of the path refinement. Let n be minimal s.t. $(p, q) \notin X_n$. We use induction on n to prove the claim.

If $n = 0$, there is a word $w \in L_{\lambda\hookleftarrow}$ such that $\min\{c(\delta^*(p, u)) \mid u \sqsubseteq w\} \neq \min\{c(\delta^*(q, u)) \mid u \sqsubseteq w\}$. Let $x, y \in c(Q)$ be the third components of $(\delta^{\lambda}_{\text{visit}})^*(\iota^{\hat{c}}_p, w)$ and $(\delta^{\lambda}_{\text{visit}})^*(\iota^{\hat{c}}_q, w)$ respectively. By Lemma 1.5.6, $x = \min\{c(\delta^*(p, u)) \mid u \sqsubseteq w\}$ and $y = \min\{c(\delta^*(q, u)) \mid u \sqsubseteq w\}$, so $x \neq y$. By definition of $\mathcal{A}^{\lambda}_{\text{visit}}$, this means that $((\delta^{\lambda}_{\text{visit}})^*(\iota^{\hat{c}}_p, w), (\delta^{\lambda}_{\text{visit}})^*(\iota^{\hat{c}}_q, w)) \notin V$.

For $n + 1 > 0$, there is a $w \in L_{\lambda\hookleftarrow}$ such that $(\delta^*(p, w), \delta^*(q, w)) \notin V$. Let $p' = \delta^*(p, w)$ and q' analogously. By induction, $(\iota^{\hat{c}}_{p'}, \iota^{\hat{c}}_{q'}) \notin V_M$. Lemma 1.5.6 tells us that there are k' and l' such that $(\delta^{\lambda}_{\text{visit}})^*(\iota^{\hat{c}}_p, w) = \iota^{k'}_{p'}$ and $(\delta^{\lambda}_{\text{visit}})^*(\iota^{\hat{c}}_q, w) = \iota^{l'}_{q'}$. Since n was chosen to be minimal, it must be true that $k' = l'$; otherwise, we would already have $p, q \notin X_0$. From Lemma 1.5.8, we know that $(\iota^{\hat{c}}_{p'}, \iota^{\hat{c}}_{q'}) \notin V_M$ if and only if $(\iota^{k'}_{p'}, \iota^{l'}_{q'}) \notin V_M$, which is false. Thus, finally, $(\iota^{\hat{c}}_p, \iota^{\hat{c}}_q) \notin V_M$. $\square$

The automaton has size $|\mathcal{A}^{\lambda}_{\text{visit}}| \in \mathcal{O}(|Q| \cdot |c(Q)|^2)$ and the computation of V_M brings the runtime up to $\mathcal{O}(|\mathcal{A}^{\lambda}_{\text{visit}}| \cdot \log |\mathcal{A}^{\lambda}_{\text{visit}}|)$.

1.2.3 Further notes

Order of equivalence classes

We can consider state reduction via path refinement as a function: $\text{PR}(\mathcal{A}, \lambda) = \mathcal{A}'$, which is a representative merge of $\mathcal{A}$ w.r.t. $\equiv^{\lambda}_{\text{PR}}$. For a congruence relation R, let $\lambda_1, \ldots \lambda_n$ be an enumeration of all the equivalence classes. To achieve even better reduction, we can simply repeat the algorithm as $f(f(\ldots f(\mathcal{A}, \lambda_1) \ldots, \lambda_{n-1}), \lambda_n)$.

The automaton displayed in figure 1.3 is an example for a case where this order matters. Assume our relation has four classes, $\{q_1, q_5\}$, $\{q_2, q_3\}$, $\{q_4\}$, and $\{q_0\}$. Equivalence classes of size 1 cannot cause any state reduction, so we focus on $\lambda_1 = \{q_1, q_5\}$ and $\lambda_2 = \{q_2, q_3\}$.

Consider $\text{PR}(\mathcal{A}, \lambda_1)$. From q_1, there is a path back to λ_1 again that only visits priority 1, namely $q_1q_3q_4q_5$. That is impossible from q_5, as the first step always leads to q_0 with priority 0. Hence, no merge will occur in this step.

$\text{PR}(\mathcal{A}, \lambda_2)$ will cause a merge: from the pair (q_2, q_3), every path back to λ_2 will either move through q_2 or q_0 and thus have the minimal priority 0. The resulting automaton is shown in figure 1.4.

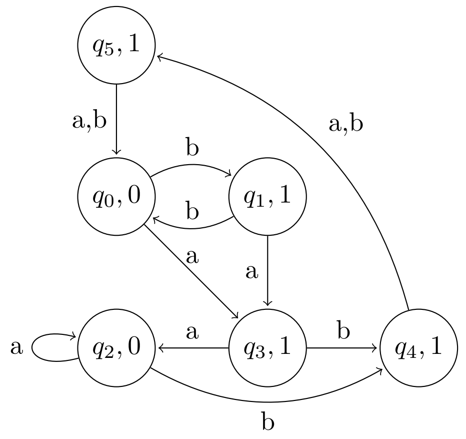

Figure 1.1: Example automaton for which the order of congruence classes matters in path refinement.

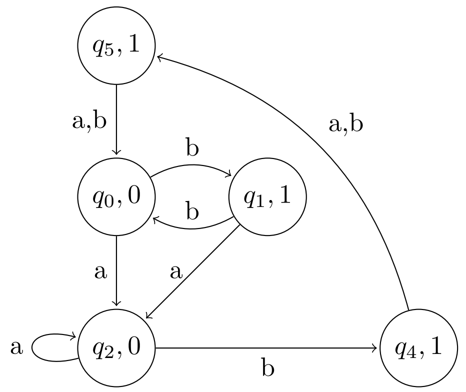

Figure 1.2: Automaton from figure 1.3 after merging q_2 and q_3.

Now look at $\mathrm{PR}(\mathrm{PR}(\mathcal{A}, \lambda_2), \lambda_1)$. As q_3 does not exist anymore, the path $q_1 q_3 q_4 q_5$ becomes $q_1 q_2 q_4 q_5$ with minimal priority of 0. In fact, now q_1 and q_5 can be merged, unlike before.

We were not able to find an easy heuristic to determine which order of classes gives the best reduction. One could of course repeat the reduction process over and over until no change is made anymore but that would bring the worst case runtime to about cubic in the number of states.

1.3 Schewe

This section is based heavily on [15] and partially adapts their results from Büchi to parity automata.

Definition 1.3.1. Let $\mathcal{A} = (Q, \Sigma, \delta, c)$ be a DPA and let $\emptyset \neq C \subseteq M \subseteq Q$. Let $\mathcal{A}' = (Q', \Sigma, \delta', c')$ be another DPA. We call $\mathcal{A}'$ a *Schewe merge of $\mathcal{A}$ w.r.t. M by candidates C* if it satisfies the following:

- There is a state $r_M \in C$ such that $Q' = (Q \setminus M) \cup \{r_M\}$.
- $c' = c \restriction_{Q'}$.
- Let $p \in Q'$ and $\delta(p, a) = q$. If $q \in M$ or if ($q \in C$ and p is not reachable from q), then $\delta'(p, a) = r_M$. Otherwise, $\delta'(p, a) = q$.

The definition of a Schewe merge is almost identical to that of a representative merge. The only difference lies therein that some additional transitions are redirected to the representative: when a transition leads to a candidate that is not in M while also moving to a different SCC.

Using the Schewe merge instead of the representative merge does not actually remove any additional states from the automaton, it only provides a better "framework" for following algorithms such as the Moore reduction. Example shows that using a Schewe merge followed by another merge of μ_M creates a better end result.

Definition 1.3.2. Let $\mathcal{A}$ be DPA with a merger function $\mu : D \to 2^Q$. For a representative merge $\mathcal{A}'$, we define the *candidate relation* $\sim_C^\mu$ as $p \sim_C^\mu q$ iff there is a $C \in \mu(D)$ with $p, q \in C$.

We say that μ is *Schewe suitable* if for all representative merges $\mathcal{A}'$, $\sim_C^\mu$ is a congruence relation, it implies language equivalence, and the reachability order restricted to any $\kappa \in \mathfrak{C}(\sim_C^\mu)$ is an equivalence relation.

Lemma 1.3.1. *Let $\mathcal{A}$ be a DPA and $\mu : D \to 2^Q$ be a merger function that is Schewe suitable. Let $\mathcal{A}'$ be a representative merge of $\mathcal{A}$ w.r.t. μ and let $\mathcal{A}''$ be the Schewe merge that uses the same choice of representatives. For all $p \sim_C^\mu q$, $\delta'(p, a) \sim_C^\mu \delta''(q, a)$.*

Proof. If $\delta''(q, a) = \delta'(q, a)$, then $\delta'(p, a) \sim_C^\mu \delta'(q, a)$ because μ is Schewe suitable and the claim is true.

Otherwise $\delta'(q, a) = q' \in \mu(M)$ for some M and q is not reachable from q'. Then $\delta''(q, a) = r_M$. By definition, $r_M \in \mu(M)$ as well. By definition of $\sim_C^\mu$, $r_m \sim_C^\mu q'$ and thus $\delta'(p, a) \sim_C^\mu \delta'(q, a) \sim_C^\mu \delta''(q, a)$. □

Lemma 1.3.2. *Let $\mathcal{A}$ be a DPA and $\mu : D \to 2^Q$ be a merger function that is Schewe suitable. Let $\mathcal{A}'$ be a representative merge of $\mathcal{A}$ w.r.t. μ and let $\mathcal{A}''$ be the Schewe merge that uses the same choice of representatives. Every run of $\mathcal{A}''$ has a suffix that is a run of $\mathcal{A}'$.*

Proof. Let $K \subseteq \mathbb{N}$ be the set of positions at which $\mathcal{A}''$ uses a transition that is not in $\mathcal{A}'$. That is, given a run ρ on α, $\rho(k + 1) \neq \delta'(\rho(k)\alpha(k))$ for all $k \in K$. If we can prove that K is finite, then that means $\rho[\max K + 1, \omega]$ is a run in $\mathcal{A}'$.

More precisely, we show that for every $\kappa \in \mathfrak{C}(\sim_C^\mu)$, there is at most one $k \in K$ such that $\rho(k + 1) \in \kappa$. Towards a contradiction assume the opposite, so there are $k < k'$ which both have this property. We label the positions in K in ascending order and have $k = k_l$ and $k' = k_{l'}$ for some $l < l'$.

Consider k_{l+1}. By definition of the Schewe merge, $\delta'(\rho(k_{l+1}), \alpha(k_{l+1})) \not\preceq^{\mathcal{A}}_{\text{reach}} \rho(k_{l+1})$. By Lemma 1.8.1, we know that $(\delta')^*(\rho(k_{l+1}), \alpha[k_{l+1}, k_{l'} + 1]) \sim^{\mu}_{\mathcal{C}} (\delta'')^*(\rho(k_{l+1}), \alpha[k_{l+1}, k_{l'} + 1]) = \rho(k_{l'} + 1)$. That means there is a state $r \in \kappa$ that is reachable from $\delta'(\rho(k_{l+1}), \alpha(k_{l+1}))$ in $\mathcal{A}$ and therefore from $\rho(k_l + 1)$ as well.

Reachability order restricted to κ is an equivalence relation, so r and $\rho(k_l + 1)$ must lie in the same SCC in $\mathcal{A}'$. That however contradicts the fact that at position k_{l+1} a redirected transition was taken. □

Lemma 1.3.3. *Let $\mathcal{A}$ be a DPA and $\mu : D \to 2^Q$ be a merger function that is Schewe suitable. Let $\mathcal{A}'$ be a representative merge of $\mathcal{A}$ w.r.t. μ that was built with representatives $R \subseteq Q$. If a Schewe merge $\mathcal{A}''$ is built with the same representatives, then $(\mathcal{A}', p) \equiv_L (\mathcal{A}'', q)$ for all $q \sim^{\mu}_{\mathcal{C}} q$.*

Proof. Let $\alpha \in \Sigma^\omega$ be some word. Let ρ' be the run of $\mathcal{A}'$ on α starting in p and let ρ'' be the run of $\mathcal{A}''$ on α starting in q. Let $k_1, \dots, k_n$ be the positions at which ρ'' uses a transition that is not present in $\mathcal{A}'$, i.e. $\rho''(k_i + 1) \neq \delta'(\rho(k_i), \alpha(k_i))$. By Lemma 1.8.2, this list must be finite.

Our goal now is to prove $\rho'(k_i + 1) \sim^{\mu}_{\mathcal{C}} \rho''(k_i + 1)$ for all i. If that is true, then $\rho'(k_n + 1) \sim^{\mu}_{\mathcal{C}} \rho''(k_n+1)$ in particular is true and therefore $\rho'(k_n+1) \equiv_L \rho''(k_n+1)$. By choice of k_n, $\rho''[k_n+1, \omega]$ is also a run in $\mathcal{A}'$ which has the same acceptance as ρ''. Since the two states are language equivalent, this is also the same acceptance as ρ'.

Assume that for some i, $\rho'(k_j + 1) \sim^{\mu}_{\mathcal{C}} \rho''(k_j + 1)$ is true for all $j < i$. As $\sim^{\mu}_{\mathcal{C}}$ is a congruence relation in $\mathcal{A}'$, $\rho'(k_i) \sim^{\mu}_{\mathcal{C}} \rho''(k_i)$ and $\rho'(k_i + 1) = \delta'(\rho'(k_i), \alpha(k_i)) \sim^{\mu}_{\mathcal{C}} \delta'(\rho''(k_i), \alpha(k_i))$. By Lemma 1.8.1, $\delta'(\rho''(k_i), \alpha(k_i)) \sim \delta''(\rho''(k_i), \alpha(k_i)) = \rho''(k_i + 1)$. □

Lemma 1.3.4. *Let $\sim$ be a congruence relation that implies language equivalence. Then $\mu^{\sim}_{skip}$ is Schewe suitable.*

Proof. Let $\mathcal{A}'$ be a representative merge of a DPA $\mathcal{A}$ w.r.t. $\mu^{\sim}_{\text{skip}}$. We prove that $p \sim q$ iff $p \sim^{\mu^{\sim}_{\text{skip}}}_{\mathcal{C}} q$. The required properties then follow from the assumptions in the statement and from Lemma 1.2.3.

We have $\text{dom}(\mu^{\sim}_{\text{skip}}) = D = \{M_\kappa \mid \kappa \in \mathfrak{C}(\sim)\}$ and $\mu^{\sim}_{\text{skip}}(M_\kappa) = C_\kappa \subseteq \kappa$. Thus, $p \sim^{\mu^{\sim}_{\text{skip}}}_{\mathcal{C}} q$ implies $p \sim q$.

On the other hand, $\kappa = C_\kappa \cup M_\kappa$, so all states of κ that remain in $\mathcal{A}'$ are C_κ and thus lie in the same equivalence class of $\sim^{\mu^{\sim}_{\text{skip}}}_{\mathcal{C}}$. □

Corollary 1.3.5. *Let $\mathcal{A}$ be a DPA and let $\sim$ be a congruence relation that implies language equivalence. For each Schewe merge $\mathcal{A}'$ of $\mathcal{A}$, $\mathcal{A} \equiv_L \mathcal{A}'$.*

Proof. Follows from Lemma 1.8.3, Lemma 1.8.4, and Theorem 1.2.5. □

We have proven that the Schewe merger function can be used to refine congruence relations (such as $\equiv_\dagger$) that, by themselves are not strong enough criteria to allow for a merging of states, to the point that they can be used for state space reduction. A final result regarding this technique is adapted from [15] and shows a relation between this algorithm and priority almost equivalence.

Lemma 1.3.6. *Let $\sim = \equiv_\dagger$ and let $\mathcal{S}'$ be a representative merge of $\mathcal{S}$ w.r.t. μ_M. There is no smaller DPA than $\mathcal{S}'$ that is priority almost equivalent to $\mathcal{A}$.*

Proof. Let $\mathcal{B}$ be a DPA that is smaller than $\mathcal{S}'$. Our goal is to show that $\mathcal{A} \not\equiv_\dagger \mathcal{B}$.

At first observe that $\mathcal{A} \equiv_\dagger \mathcal{S}'$: $\mathcal{A}$ and $\mathcal{S}$ are priority almost equivalent as for every state in $\mathcal{A}$, there is an equivalent representative in $\mathcal{S}$. $\mathcal{S}$ and $\mathcal{S}'$ are Moore equivalent by Lemma **??** and thus they are also priority almost equivalent by Lemma **??**.

Assume that $\mathcal{S}' \equiv_\dagger \mathcal{B}$ holds, so for all states in $\mathcal{S}'$, there is a a priority almost equivalent state in $\mathcal{B}$. We define a function f that maps to each equivalence class of $\equiv_\dagger$ in $\mathcal{S}'$ all states in $\mathcal{B}$ that are equivalent to it, i.e. if $\kappa \subseteq Q_{\mathcal{S}'}$ is an equivalence class, then $f(\kappa) = \{q \in Q_\mathcal{B} \mid \exists p \in \kappa : (\mathcal{S}', p) \equiv_\dagger (\mathcal{B}, q)\}$. Note that $f(\kappa)$ can never be empty by the assumption of $\mathcal{S}' \equiv_\dagger \mathcal{B}$.

As $\mathcal{B}$ is smaller than $\mathcal{S}'$, the pigeonhole principle applies and we can fix κ to be one equivalence class such that $|f(\kappa)| < |\kappa|$.

By Lemma **??**, there is an SCC C in $\mathcal{S}'$ that contains all states in κ. Without loss of generality we can assume that likewise there is an SCC D in $\mathcal{B}$ that contains $f(\kappa)$. If no such SCC would exist, we could simply apply the Schewe merger to $\mathcal{B}$ to find an automaton that is smaller than $\mathcal{S}'$ and does have this property.

C and D must be non-trivial SCCs: if C would be trivial, κ would contain only one element and $f(\kappa)$ would be empty. If D would be trivial, $f(\kappa) = \{q\}$ would contain of only one state. Since $\mathcal{S}' \equiv_\dagger \mathcal{B}$, there is a state $p \in \kappa \subseteq C$ in $\mathcal{S}'$ with $(\mathcal{S}', p) \equiv_\dagger (\mathcal{B}, q)$. Since C is not trivial, there is a word w from which $\mathcal{S}'$ moves from p back to p. $\mathcal{B}$ however, leaves $f(\kappa)$ with that word, as D is trivial, so q cannot reach itself again. This is a contradiction, as $\equiv_\dagger$ is a congruence relation.

We claim that there is a state $p \in \kappa$ such that there is a family of words $(w_q)_{q \in Q_\mathcal{B}}$ such that $\mathcal{S}'$ does not leave C reading these words from p and $c_{\mathcal{S}'}(\delta^*_{\mathcal{S}'}(p, w)) \neq c_\mathcal{B}(\delta^*_\mathcal{B}(q, w))$. (in other words, w_q is a witness for p and q being not Moore-equivalent.)

Towards a contradiction, assume that the claim is false and that for every $p \in \kappa$, there is a state $q_p \in f(\kappa)$ that does not satisfy the property. As $|\kappa| > |f(\kappa)|$ we can again use the pigeonhole principle and obtain two states $p_1, p_2 \in \kappa$ such that $q_{p_1} = q_{p_2}$. We call this state $q := q_{p_1}$.

For each word $w \in \Sigma^*$, $c^*_{\mathcal{S}'}(p_1, w) = c^*_\mathcal{B}(q, w)$ or $\mathcal{S}'$ leaves C while reading w from p_1. The same holds for p_2. If for all w the first case would apply, then $c^*_{\mathcal{S}'}(p_1, w) = c^*_{\mathcal{S}'}(p_2, w)$ for all w and thus $p_1 \equiv_M p_2$. This is impossible, as the two states would have been merged in the construction of $\mathcal{S}'$. Without loss of generality, we assume that p_1 breaks the pattern and there is a w such that $c^*_{\mathcal{S}'}(p_1, w) \neq c^*_\mathcal{B}(q, w)$ but $\mathcal{S}'$ leaves C when reading w from p_1. Let this w have minimal length.

Let ρ_1 and ρ_2 be the respective runs of $\mathcal{S}$ (note that we are using the automaton here that is not yet Moore-minimized) on w from p_1 and p_2. At some point k, ρ_1 leaves C.

We can use the claim that he have just shown to finish our overall proof. Fix an arbitrary $q_0 \in f(\kappa)$. We define a sequence of finite words $(\alpha_n)_{n \in \mathbb{N}}$ such that every α_n is a prefix of α_{n+1} and the runs of $\mathcal{S}'$ and $\mathcal{B}$ from p and q_0 respectively differ in priority at least n times. Then $\alpha := \bigcup_n \alpha_n$ is an ω-word that is a witness for $(\mathcal{S}', p)$ and $(\mathcal{B}, q_0)$ not being priority almost equivalent.

We make sure that after reading any α_n from p, $\mathcal{S}'$ moves back to p. Let $\alpha_0 := \varepsilon$ and assume for induction that α_n has already been defined. After reading α_0, $\mathcal{S}'$ reaches p and $\mathcal{B}$ reaches some q.

If $q \notin f(\kappa)$, i.e. $(\mathcal{S}', p) \not\equiv_{\dagger} (\mathcal{B}, q)$, there is a witness β and we can simply set $\alpha := \alpha_n \beta$. Otherwise, $q \in f(\kappa)$. With the claim we have proven, we can find a word w_q such that reading the word from p and q leads to states p' and q' that have different priorities but p' is still in C. Thus, there is a word u that leads back from p' to p. We then set $\alpha_{n+1} := \alpha_n w_q u$. This finishes our proof. $\square$

Bibliography

[1] Filippo Bonchi and Damien Pous. 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, 2013. ACM.

[2] Julius Richard Büchi. On a decision method in restricted second order arithmetic. 1966.

[3] Olivier Carton and Ramón Maceiras. Computing the rabin index of a parity automaton. *RAIRO - Theoretical Informatics and Applications - Informatique Théorique et Applications*, 33(6):495–505, 1999.

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

[5] Kousha Etessami, Thomas Wilke, and Rebecca A. Schuller. Fair simulation relations, parity games, and state space reduction for büchi automata. In *Automata, Languages and Programming*, pages 694–707, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[6] Carsten Fritz and Thomas Wilke. Simulation relations for alternating büchi automata. *Theor. Comput. Sci.*, 338(1-3):275–314, June 2005.

[7] Monika Rauch Henzinger and Jan Arne Telle. Faster algorithms for the nonemptiness of streett automata and for communication protocol pruning. In *Algorithm Theory — SWAT'96*, pages 16–27, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[8] John Hopcroft. 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, 01 1971.

[9] Tao Jiang and B. Ravikumar. Minimal nfa problems are hard. In *Automata, Languages and Programming*, pages 629–640, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.

[10] Richard Mayr and Lorenzo Clemente. Advanced automata minimization. In *POPL 2013*, Oct 2012.

[11] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. 1943. *Bulletin of mathematical biology*, 52 1-2:99–115; discussion 73–97, 1990.

[12] Edward F. Moore. Gedanken-experiments on sequential machines. In Claude Shannon and John McCarthy, editors, *Automata Studies*, pages 129–153. Princeton University Press, Princeton, NJ, 1956.

[13] Nir Piterman. From nondeterministic büchi and streett automata to deterministic parity automata. *Logical Methods in Computer Science*, 3(3), 2007.

[14] Michael O Rabin and Dana Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3:114–125, 04 1959.

[15] Sven Schewe. Beyond Hyper-Minimisation—Minimising DBAs and DPAs is NP-Complete. In Kamal Lodaya and Meena Mahajan, 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, 2010. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

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

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