0.1 Introduction

Finite automata are a long established computation model that dates back to sources such as [10] and [13]. 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 [7]. Regarding nondeterministic finite automata (NFA), [8] 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 [14] and spawned heuristic approaches such as [14], [9], or [5].

As [16] 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 ([11]), that use a parity function rather than the usual acceptance set. [12] 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 ([14]).

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 **??**.
- Delayed simulation equivalence with resets, $\equiv_{\mathrm{deR}}$. Defined in section **??**.
- Path refinement equivalence, $\equiv_{\mathrm{PR}}$. Defined in
- Threshold Moore equivalence, $\equiv_{\mathrm{TM}}$. Defined in

- Labeled SCC filter equivalence, $\equiv_{\mathrm{LSF}}$. Defined in

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 [6].

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 [7]. 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 $D \subseteq 2^Q$. If all sets in D are pairwise disjoint and for all $X \in D$, $\mu(X) \subseteq X$, we call μ a *merger function*.

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 $\rho(i + 1) \sim q \sim \rho'(i + 1)$. □

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 **??**.
- Delayed simulation merger, μ_{de}. Defined in section **??**.

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 $c(p) = c(q)$. Let $\mathcal{A}'$ be a representative merge of $\mathcal{A}$ w.r.t. $\mu^{\sim}_{\div}$. For all $q \in Q'$, $(\mathcal{A}, q) \equiv_M (\mathcal{A}', q)$.*

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. □

Lemma 1.1.13. *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. By Lemma 1.1.12, we have $(\mathcal{A}, q) \equiv_M (\mathcal{A}', q)$ for all $q \in Q'$. On the other hand, for all $q \in Q \setminus Q'$, there is a representative $r_{[q]_\sim} \in Q'$. We have $q \sim r_{[q]_\sim}$ and thus $q \equiv_M r_{[q]_\sim}$ and therefore also $(\mathcal{A}, q) \equiv_M (\mathcal{A}, r_{[q]_\sim}) \equiv_M (\mathcal{A}', r_{[q]_\sim})$. □

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 1.1.14. *$\preceq^{\mathcal{S}}_{reach}$ 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 1.1.15. *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 [15], 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 1.1.16. *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(𝒜)
2:     c′ : Q → ℕ, q ↦ c(q)
3:     M(𝒜, c′)
4:     return c′
5: end function

6: function M(𝒜 ↾_P, c′)
7:     if P = ∅ then
8:         return 0
9:     end if
10:    min ← 0
11:    for SCC S in 𝒜 ↾_P do
12:        m := min c(S) mod 2
13:        X := c^{-1}(m)
14:        for q ∈ X do
15:            c′(q) ← m
16:        end for
17:        S′ := S \ X
18:        m′ ←M(𝒜 ↾_{S′}, c′)
19:        if m′ even then
20:            if m even then
21:                δ := m
22:            else
23:                δ := m − 2
24:            end if
25:        else
26:            δ := m − 1
27:        end if
28:        for q ∈ S′ do
29:            c′(q) ← c′(q) − δ
30:        end for
31:        min ← min{min, m}
32:    end for
33:    return min
34: end function
```

1.2 Threshold Moore

Definition 1.2.1. Let $\mathcal{A} = (Q, \Sigma, \delta, c)$ be a DPA. For a set $K \subseteq \mathbb{N}$, we define the *priority threshold* of $\mathcal{A}$ as $\mathcal{T}(\mathcal{A}, K) := (Q, \Sigma, \delta, c')$ with $c'(q) = \begin{cases} c(q) & \text{if } c(q) \notin K \\ \max c(Q) + 1 & \text{else} \end{cases}$.

For a relation $\sim$, we define $\mathcal{T}(\sim, K) := \approx$ as $(\mathcal{A}, p) \approx (\mathcal{B}, q)$ iff $(\mathcal{T}(\mathcal{A}, K), p) \sim (\mathcal{T}(\mathcal{B}, K), q)$.

Lemma 1.2.1. *If $\sim$ is an equivalence relation, then so is $\mathcal{T}(\sim, K)$. If $\sim$ is a congruence relation, then so is $\mathcal{T}(\sim, K)$.*

Proof. □

Definition 1.2.2. We set $\equiv_M^{\leq k} := \mathcal{T}(\equiv_M, \{n \in \mathbb{N} \mid n > k\})$.

Definition 1.2.3. Let $\mathcal{A}$ and $\mathcal{B}$ be DPAs and let $\sim$ be an equivalence relation that implies language equivalence. We define a relation $\equiv_{\mathrm{TM}}^{\sim}$ such that $(\mathcal{A}, p) \equiv_{\mathrm{TM}}^{\sim} (\mathcal{B}, q)$ if and only if all of the following are satisfied:

1. $c_1(p) = c_2(q)$
2. $(\mathcal{A}, p) \equiv_M^{\leq c(p)} (\mathcal{B}, q)$
3. $(\mathcal{A}, p) \sim (\mathcal{B}, q)$

We can notice that merging classes of $\equiv_{\mathrm{TM}}^{\sim}$ in a specific order is a valid language-preserving operation. This is then used to define a fitting merger function. For this next part, let $\mathcal{A}$ be a DPA and let $\kappa \in \mathfrak{C}(\equiv_{\mathrm{TM}}^{\sim})$ be a set of states in $\mathcal{A}$. By definition of $\equiv_{\mathrm{TM}}^{\sim}$, all states in this class have a unique priority k. Let $\mathcal{A}'$ be a representative merge of $\mathcal{A}$ w.r.t. κ.

Lemma 1.2.2. *For all $q \in Q'$, $(\mathcal{A}, q) \sim (\mathcal{A}', q)$.*

Proof. Let ρ and ρ' be the runs of $\mathcal{A}$ and $\mathcal{A}'$ on $\alpha \in \Sigma^\omega$ starting from q. We claim that ρ is accepting iff ρ' is accepting.

By Lemma 1.1.11, $(\mathcal{A}, \rho(i)) \equiv_L (\mathcal{A}, \rho'(i))$ and $(\mathcal{A}, \rho(i)) \equiv_M^{\leq k} (\mathcal{A}, \rho'(i))$ for all i. Now there are two cases: if $c(\rho)$ sees infinitely many priorities of at most k, then $c(\rho')$ sees the same priorities at the same positions and thus $\min \mathrm{Inf}(c(\rho)) = \min \mathrm{Inf}(c(\rho'))$. By definition of a representative merge, $c(\rho') = c'(\rho')$.

Otherwise there is a position n from which $c(\rho)$ only is greater than k and therefore the same is true for $c(\rho')$. That means reading $\alpha[n, \omega]$ from $\rho'(n)$ in either $\mathcal{A}$ or $\mathcal{A}'$ yields the same run which has the same acceptance as ρ. □

Lemma 1.2.3. *For all $q \in Q'$ with $c(q) \leq k$, $(\mathcal{A}, q) \equiv_M^{\leq k} (\mathcal{A}', q)$.*

Proof. Let ρ and ρ' be the runs of $\mathcal{A}$ and $\mathcal{A}'$ on $\alpha \in \Sigma^\omega$ starting from q. We claim that ρ is accepting iff ρ' is accepting.

By Lemma 1.1.11, $(\mathcal{A}, \rho(i)) \equiv_M^{\leq k} (\mathcal{A}, \rho'(i))$ for all i which especially means that for all i, $c(\rho(i)) =^{\leq k} c(\rho'(i))$. Since $c(\rho'(i)) = c'(\rho'(i))$, that also implies $c(\rho(i)) =^{\leq k} c'(\rho'(i))$ which means that $(\mathcal{A}, q) \equiv_M^{\leq k} (\mathcal{A}', q)$. □

Lemma 1.2.4. *For all $q \in Q'$ with $c(q) \leq k$, $(\mathcal{A}, q) \equiv^{\sim}_{TM} (\mathcal{A}', q)$.*

Proof. Representative merges never change priorities assigned to states. Together with Lemma 1.2.2 and Lemma 1.2.3, this already finishes the proof. □

Lemma 1.2.5. *For all $p, q \in Q'$ with $\min\{c(p), c(q)\} \leq k$, $(\mathcal{A}, p) \equiv^{\sim}_{TM} (\mathcal{A}, q)$ iff $(\mathcal{A}', p) \equiv^{\sim}_{TM} (\mathcal{A}', q)$.*

Proof. If $c(p) = c(q) \leq k$: By Lemma 1.2.4, we know that $(\mathcal{A}, p) \equiv^{\sim}_{\mathrm{TM}} (\mathcal{A}', p)$ and $(\mathcal{A}, q) \equiv^{\sim}_{\mathrm{TM}} (\mathcal{A}', q)$. If now $(\mathcal{A}, p) \equiv^{\sim}_{\mathrm{TM}} (\mathcal{A}, q)$ holds, then also $(\mathcal{A}', p) \equiv^{\sim}_{\mathrm{TM}} (\mathcal{A}', q)$ and vice versa.

If $c(p) \neq c(q)$, then also $c'(p) \neq c'(q)$ and we have both $(\mathcal{A}, p) \not\equiv^{\sim}_{\mathrm{TM}} (\mathcal{A}, q)$ and $(\mathcal{A}', p) \not\equiv^{\sim}_{\mathrm{TM}} (\mathcal{A}', q)$. □

Definition 1.2.4. Let $\mathcal{A}$ be a DPA and let $\sim$ be an equivalence relation that implies language equivalence. We define the *Threshold Moore merger function* $\mu^{\sim}_{\mathrm{TM}} := \mu_{\div}^{\equiv^{\sim}_{\mathrm{TM}}}$.

Theorem 1.2.6. *Let $\mathcal{A}$ be a DPA and let $\sim$ be an equivalence relation that implies language equivalence. Every representative merge of $\mathcal{A}$ w.r.t. $\mu^{\sim}_{TM}$ is language equivalent to $\mathcal{A}$.*

Proof. Let $\kappa_1, \ldots, \kappa_m$ be an enumeration of $\mathfrak{C}(\equiv^{\sim}_{\mathrm{TM}})$ such that the (unique) priority of states in κ_i is k_i and $k_1, \ldots, k_m$ is a descending series.

If we build a representative merge of $\mathcal{A}$ w.r.t. κ_1 by candidates $_1$, we obtain some DPA $\mathcal{A}_1$. By Lemma 1.2.5, for all $i > 1$, the states in κ_i are still pairwise $\equiv^{\sim}_{\mathrm{TM}}$ equivalent. Moreover, κ_i is an $\equiv^{\sim}_{\mathrm{TM}}$ equivalence class in $\mathcal{A}_1$.

That means we can continue building representative merges in order of the enumeration and our previous results apply. In the end, we obtain a DPA $\mathcal{A}'$ that is language equivalent to $\mathcal{A}$ by Lemma 1.2.2. □

1.3 Labeled SCC Filter

Definition 1.3.1. Let $\mathcal{A} = (Q, \Sigma, q_0, \delta, c)$ be a DPA. We define $\mathcal{A} \restriction^c_{=k} := \mathcal{A} \restriction_P$ with $P = \{q \in Q \mid c(q) = k\}$. Analogously, we define $\mathcal{A} \restriction^c_{>k}$.

We define a relation $R_k \subseteq Q \times Q$ such that $(p, q) \in R_k$ if and only if all of the following are true:

1. $\min\{c(p), c(q)\} > k$
2. $p \equiv_L q$
3. $p \equiv^{\leq k}_M q$
4. In $\mathcal{A} \restriction^c_{>k}$, p and q lie in different SCCs.

We define $\equiv^k_{\mathrm{LSF}} \subseteq Q \times Q$ to be the reflexive and transitive closure of R_k.

Lemma 1.3.1. *$\equiv^k_{LSF}$ is an equivalence relation.*

Definition 1.3.2. Let $\mathcal{A}$ be a DPA and $k \in \mathbb{N}$. We define $\preceq_k \subseteq Q \times Q$ to be a total extension of the reachability preorder in $\mathcal{A} \restriction^c_{\geq k}$.

Let λ be an equivalence class of $\equiv^k_{\mathrm{LSF}}$. Let $r \in \lambda$ be a representative of λ that is $\preceq_k$-maximal. We set $\lambda' := \{q \in \lambda \mid q \prec_k r\} \cup \{r\}$. We call an automaton $\mathcal{A}'$ a *LSF^k_λ-merge* of $\mathcal{A}$ if it is a representative merge of $\mathcal{A}$ w.r.t. λ' that uses the representative $r_{\lambda'} = r$.

Theorem 1.3.2. *Let $\mathcal{A}$ be a DPA and let $\mathcal{A}'$ be a LSF^k_λ-merge of $\mathcal{A}$. Then $L(\mathcal{A}) = L(\mathcal{A}')$.*

Proof. Let r_λ be the representative that is used in the construction of $\mathcal{A}'$. Let $q \in Q'$ be a state in the representative merge and let $\alpha \in \Sigma^\omega$. Let ρ and ρ' be the runs of $\mathcal{A}$ and $\mathcal{A}'$ on α starting from q. We claim that ρ is accepting iff ρ' is accepting.

By Lemma **??**, we know that $\rho(i) \equiv_L \rho'(i)$ and $\rho(i) \equiv^{\leq k}_M \rho'(i)$ for all i. If there is a position n from which on $\rho'[n, \omega]$ is both a valid run in $\mathcal{A}$ and $\mathcal{A}'$, then we know that ρ is accepting if and only if ρ' is accepting since $\rho(n) \equiv_L \rho'(n)$.

If ρ' visits infinitely many states with priority equal to or less than k, then ρ and ρ' share the same minimal priority that is visited infinitely often and thus have the same acceptance.

For the last case, assume that ρ' uses infinitely many redirected edges but from some point n_1 on stays in $\mathcal{A} \restriction^c_{>k}$. Let $n_3 > n_2 > n_1$ be the next two positions at which ρ' uses a redirected edge, i.e. $\delta(\rho'(n_2), \alpha(n_2)) \neq \delta'(\rho'(n_2), \alpha(n_2))$ and analogous for n_3. Note that $\delta'(\rho'(n_2), \alpha(n_2)) = \delta'(\rho'(n_3), \alpha(n_3)) = r_\lambda$, since all redirected transition target the representative state. Let we call $\delta(\rho'(n_3), \alpha(n_3)) = q$. Since between n_2 and n_3 no redirected transition is taken, $\rho'[n_2, n_3]$ is a valid path in $\mathcal{A}$, so we have $r_\lambda \preceq_k q$ by choice of n_1. The fact that transitions to q are redirected to r_λ however requires that $q \prec_k r_\lambda$, which would be a contradiction. □

Lemma 1.3.3. *Let $\mathcal{A}$ be a DPA and let $\mathcal{A}'$ be a LSF^k_λ-merge of $\mathcal{A}$. Let $\equiv^l_{LSF}$ be the LSF-relation in $\mathcal{A}$ and let $\equiv^l_{LSF'}$ be the LSF-relation in $\mathcal{A}'$. If $l \leq k$, then $\equiv^l_{LSF} \restriction_{Q' \times Q'} \supseteq \equiv^l_{LSF'}$.*

Proof. Let R_l and R'_l be the relations used in the definition of $\equiv^l_{\mathrm{LSF}}$ and $\equiv^l_{\mathrm{LSF}'}$. We prove $R'_l \subseteq R'_l \restriction_{Q' \times Q'}$. If that is true, then so is the statement of our Lemma. We do so by considering the four properties of R_l individually.

The first point is clear; $c' = c \restriction_{Q'}$, so $c'(p) = c(p)$ and $c'(q) = c(q)$.

For the second point, consider Theorem 1.3.2. By making p or q the initial state of our DPA, we observe that neither state has its language changed by the construction, so they must still be equal.

For the third point, let $\equiv_M^{\lhd l}$ be the l-threshold Moore equivalence in $\mathcal{A}'$. Let $w \in \Sigma^*$ be an arbitrary word, $p \equiv_M^{\leq l} p'$, $p' := (\delta')^*(p, w)$, and $q' := (\delta')^*(q, w)$. Using Lemma **??**, we know that, since $p \equiv_M^{\leq l} q$, also $p' \equiv_M^{\leq l} q'$. In particular, this means $c(p') =^{\leq l} c(q')$. As w was chosen to be arbitrary, that means $p \equiv_M^{\lhd l} q$.

Lastly, for the fourth point, assume that there are states p, q which lie in different SCCs in $\mathcal{A} \restriction^c_{>l}$ but not in $\mathcal{A}' \restriction^c_{>l}$. Without loss of generality, we assume that in $\mathcal{A} \restriction^c_{>l}$, p is not reachable from q. In $\mathcal{A}' \restriction^c_{>l}$ however, this is possible, so let ρ' be a path from q to p. We can assume ρ' to pay exactly one visit to λ; there has to be at least one visit, as otherwise the path would also be available in $\mathcal{A}$; if there would be multiple visits, all of them would end at r_λ, so we could cut those parts from the run. Let uv be words that induce that run, i.e. $\delta^*(q, u) \in \lambda$ and $\delta^*(r_\lambda, v) = p$.

We distinguish two cases. In the first case, q is reachable from p in $\mathcal{A}$ by some word w. Here, consider reading the word vwu from r_λ in $\mathcal{A}$. The run moves to p by v, then to q by w, then to $\delta^*(q, u) \in \lambda$. $\delta^*(q, u)$ was the state from which the redirected transition was taken in ρ', so it cannot be reachable from r_λ by definition of the merge. This is a contradiction.

For the second case, q is not reachable from p in $\mathcal{A}$. Since the two states lie in a common SCC in $\mathcal{A}'$ however, there is a path π' from p to q. With the same argument as before, we can assume that π' leads to r_λ via some word u' and from there to q via some v'. As in the first case, the word $v'u$ gives us a path from r_λ to $\delta^*(q, u)$ which is a contradiction. □

The two previous statements provide us with a possible algorithm to perform state space reduction with the LSF method. Starting at $k = \min c(Q) - 1$ and iterating up to $\max c(Q)$, compute $\equiv^k_{\mathrm{LSF}}$ and build representative merges of each equivalence class. By Lemma 1.3.3, strictly iterating once through all k in ascending order gives us all possible merges.

The final question that remains is how to compute $\equiv^k_{\mathrm{LSF}}$ itself. This can be done rather easily

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] 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.

[7] 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.

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

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

[10] 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.

[11] 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.

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

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

[14] 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.

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

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