State Space Reduction For Parity Automata

Christof Löding
RWTH Aachen University, Germany
loeding@informatik.rwth-aachen.de

Andreas Tollkötter
RWTH Aachen University, Germany
andreas.tollkoetter@rwth-aachen.de

Abstract

Exact minimization of ω-automata is a difficult problem and heuristic algorithms are a subject of current research. We propose several new approaches to reduce the state space of deterministic parity automata. These are based on extracting information from structures within the automaton, such as strongly connected components, coloring of the states, and equivalence classes of given relations, to determine states that can safely be merged. We also establish a framework to generalize the notion of quotient automata and uniformly describe such algorithms. The description of these procedures consists of a theoretical analysis as well as data collected from experiments.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory; Theory of computation → Automata over infinite objects; Theory of computation → Regular languages

Keywords and phrases automata, ω-automata, parity, minimization, state space reduction, deterministic, simulation relations

Digital Object Identifier 10.4230/LIPIcs.CSL.2020.27

Related Version `https://github.com/atollk/master-thesis/blob/master/tex/thesis.tex`

Supplement Material https://github.com/atollk/master-thesis

1 Introduction

Finite automata on ω-words (one sided infinite words) have been introduced in [2] as a formalism for a deciding a logical theory. Since then, such automata have turned out to be a useful tool in verification of finite state-based systems. In particular, nondeterministic Büchi automata (NBA) are a standard tool in model checking for expressing properties of non-terminating systems, see [1]. In some applications, there are algorithms that require the property to be represented by a deterministic automaton, like model checking of probabilistic systems (see, e.g., [1, Section 10.3]), or synthesis of finite state systems from ω-regular specifications (see [20] for an overview of the theory, and [12] for recent developments in practice). Deterministic ω-automata require a more expressive acceptance condition than nondeterministic Büchi automata in order to capture the same language class. One such condition that is widely used because of its compact representation and its good algorithmic properties is the parity condition that dates back to [13] (see the surveys [19, 21] on the theory of ω-automata). In a parity automaton, each state is assigned a priority, which is a natural number. We use here the convention that a run is accepting if the smallest priority that is seen infinitely often is even.

We consider here the problem of finding algorithms for reducing the state space of deterministic parity automata (DPA). Such reduction algorithms can be used as a post processing step that are applied after a determinization construction, and before the DPAs are used in further algorithms.

While deterministic finite automata on finite words can be minimized very efficiently [9] by merging language equivalent states, the problem becomes NP-hard even for deterministic Büchi automata [17], which are the special case of DPAs using only priorities 0 and 1. While language equivalence of states in DPAs can be computed in polynomial time by a simple adaption of emptiness for Streett automata (see the "fair state problem" in [5]), merging language equivalent states of DPAs does, in general, not preserve the language. Heuristic approaches for reducing the state space of ω-automata, usually based on simulation relations, have up to now mainly focused on NBAs, e.g., [18, 6, 11], or even on alternating Büchi or parity automata [7, 8].

Because of the applications of DPAs in synthesis and probabilistic model checking, we think that it is worth studying the problem of state space reduction also for DPAs. Typically, state space reduction is done by identifying classes of equivalent states that can be merged, and then constructing the quotient automaton in case of a congruence relation, or redirecting all incoming transitions of a class to a representative of that class, and deleting all other states. The most basic merge for DPAs is obtained by interpreting a DPA as a Moore automaton with the priorities as output, which can then be minimized efficiently by merging states that produce the same output for every input sequence [9]. In this context, we call the equivalence relation which considers two states to be equivalent if they are merged by this algorithm the *Moore equivalence*.

While we also take the basic approach of computing states that can be merged, we sometimes need to be careful in the selection of representatives. For that reason, we introduce the notion of "merger templates", which map sets of states in the original DPA, called the merge set, to other sets of states, called the candidate set. The easiest interpretation, which we refer to as representative merge, allows us to merge all states from the merge set into any single representative that is chosen from the candidate set.

We formulate some basic known reduction techniques for DPAs in this framework. Furthermore, we analyze the known notion of delayed simulation for DPAs. Delayed simulation has been introduced in [6] for nondeterministic Büchi automata. In [8] the notion has been extended to alternating parity automata, but it is shown there that quotienting alternating (and also nondeterministic) parity automata w.r.t. delayed simulation does not preserve the language. For this reason, [8] introduces variants of delayed simulation that can be used for merging. We revisit the definition of delayed simulation and show that for DPAs the corresponding quotient preserves the language.

As our main contribution, we propose three new equivalence relations that can be used for merging states in DPAs, which we call *path refinement*, *threshold Moore*, and *labeled SCC filter (LSF)*. All these techniques require a given equivalence relation $\sim$ over the state space that implies language equivalence of states. This equivalence relation has to be computed separately (it can be the full language equivalence relation, or just a subset of it). In our experiments, we use a relation that is produced as a by-product of the determinization construction.

If $\sim$ is a congruence (like full language equivalence), then path refinement refines one of the congruence classes up to a point such that the remaining blocks can be merged. In the threshold Moore technique, one computes the Moore equivalence of states, considering only the priorities less than or equal to some k, and intersects this with $\sim$. All states of priority k

that are equivalent in this intersection can be merged. Finally, the LSF merger template removes states also based on the Moore equivalence up to k, but it merges states that are in different SCCs of the DPA after removing all states up to priority k.

We illustrate all these new techniques on small examples, exhibit efficient algorithms for computing the corresponding relations, and provide some experimental data showing that they can achieve significant reductions on DPAs obtained from specifications from the competition SYNTCOMP [10].

The remainder of this paper is structured as follows. In Section 2 we give basic definitions and introduce the notion of merger template. In Section 3, we revisit the notion of delayed simulation. In Sections 4–6 we present our three new approaches. The experimental evaluation is given in Section 7, and in Section 8 we conclude.

2 Automata and Merger Templates

We consider deterministic parity automata (DPA), which are, syntactically, a specific type of Moore automaton. A Moore automaton is of the form $\mathcal{A} = (Q, \Sigma, \delta, f)$ with a finite set Q of states, the input alphabet Σ, a transition function $\delta : Q \times \Sigma \to Q$, and an output function $f : Q \to \Gamma$ for some output alphabet Γ. Note that we define the automaton without initial state because we are interested in reducing the number of states of automata by computing equivalence relations on states. In this context, the initial state does not play any role.

We use the standard notations Σ^* for all finite words $w = a_0 a_1 \cdots a_n$ with all $a_i \in \Sigma$, and Σ^ω for the set of infinite words $\alpha = a_0 a_1 a_2 \cdots$ with all $a_i \in \Sigma$. We write $\alpha(i)$ for the ith letter a_i of α. When estimating the complexity of algorithms, we assume that $|\Sigma|$ is a constant.

A run of $\mathcal{A}$ from state $q_0 \in Q$ on an infinite input word $\alpha \in \Sigma^\omega$ is an infinite state sequence $\rho = q_0 q_1 q_2 \cdots \in Q^\omega$ such that $\delta(q_i, \alpha(i)) = q_{i+1}$ for all i. The generated output of $\mathcal{A}$ on α starting from q_0 is the sequence $f(\rho) = f(q_0) f(q_1) \cdots$ of outputs at the states in the run. Similarly, one defines runs and outputs for finite input words. As usual, we write $\delta^*(q, w)$ for the state that is reached by the run on w that starts in q.

A DPA is a special Moore automaton $\mathcal{A} = (Q, \Sigma, \delta, c)$, where the output function is of the form $c : Q \to \mathbb{N}$, and is called the priority function. For $q \in Q$, we refer to $c(q)$ as the priority of q, and for $P \subseteq Q$, we let $c(P) = \{c(q) \mid q \in P\}$. A run ρ of a DPA is called accepting if in $c(\rho)$ the smallest priority that occurs infinitely often is even. The word $\alpha \in \Sigma^\omega$ is accepted from $q \in Q$ if the run of $\mathcal{A}$ on α from q is accepting. We write $L(\mathcal{A}, q)$ for the set of all words accepted by $\mathcal{A}$ from q.

In the remainder of the paper, $\mathcal{A}$ with the above components is always a DPA if not noted otherwise.

We consider several types of different relations, mostly over the state domain Q. A relation R is a preorder if it is reflexive and transitive. R is an equivalence relation if it is a symmetric preorder. R is a congruence relation if it is an equivalence relation that is compatible with δ, i.e., if $(p, q) \in R$, then also $(\delta(p, a), \delta(q, a)) \in R$ for all $a \in \Sigma$.

If $\sim$ is an equivalence relation and $\mathcal{A}$ is a DPA, we write $\mathfrak{C}(\sim) \subseteq 2^Q$ for the set of equivalence classes in $\mathcal{A}$. We define two basic equivalence relations that are used throughout the paper.

▶ **Definition 1.** *The* language equivalence relation *is defined by* $p \equiv_L q$ *iff* $L(\mathcal{A}, p) = L(\mathcal{A}, q)$.

The Moore equivalence relation *is defined by* $p \equiv_M q$ *iff* $c(\delta^*(p, w)) = c(\delta^*(q, w))$ *for all finite words* $w \in \Sigma^*$ *(that is, for every input word, the sequence of priorities when starting in* p *is the same as the one when starting in* q*).*

Both of these relations are actually congruence relations. It is well known that merging language equivalent states does not preserve the accepted language in general. Consider, for example, the DPA from Figure 4 on page 10. All three states are language equivalent, accepting the words with finitely many c and infinitely many a. But it is not possible to merge any of the states as that would change the languages of the remaining states.

In contrast, Moore equivalent states can be merged without changing the language. The main aim of this paper is to identify other conditions under which language equivalent states can be merged. We say that a relation $\sim$ implies language equivalence if $p \sim q$ implies that $p \equiv_L q$.

2.1 Merger Templates

The merge operations that we use are more general than quotient automata. Consider, for example, the DPA in Figure 5 on page 12. As we explain in Section 6, it is possible to remove the states q_1, q_2, and to redirect the incoming transitions of these states to q_3 or q_4 instead. We say that $M = \{q_1, q_2\}$ is a merge set, and that $C = \{q_3, q_4\}$ is the corresponding candidate set.

We define the notion of a merger template, which maps a collection of such merge sets to their corresponding candidate sets, and the notion of representative merge, which merges the states in the merge sets into a single candidate, respectively.

▶ **Definition 2.** *Let $\mu : D \to (2^Q \setminus \{\emptyset\})$ be a function for some $D \subseteq 2^Q$. We call μ a* merger template *if all sets in D are pairwise disjoint and for all sets $M \in D$, $\mu(M) \cap (\bigcup D \setminus M) = \emptyset$. The latter condition means that the candidates $\mu(M)$ for M cannot be inside any other merge set (but they can be inside M).*

A representative merge *$\mathcal{A}'$ of $\mathcal{A}$ w.r.t. μ is constructed by choosing a representative $r_M \in \mu(M)$ for all $M \in D$ and then removing all states in $M \setminus \{r_M\}$. Transitions that originally lead to one of the removed states are redirected to the representative r_M instead.*

The notion of quotient automaton w.r.t. a congruence relation is captured by a representative merge for the merger template that maps each congruence class to itself. We illustrate this on the example of Moore equivalence.

▶ **Definition 3.** *The* Moore merger template *is defined as $\mu_M : \mathfrak{C}(\equiv_M) \to 2^Q$ with $\mu_M(\kappa) = \kappa$ for each $\kappa \in \mathfrak{C}(\equiv_M)$.*

Then, the following is an easy consequence of the definitions.

▶ **Proposition 4.** *A representative merge of a DPA w.r.t. μ_M is language equivalent to the original and isomorphic to the quotient automaton w.r.t. $\equiv_M$.*

When we apply merge operations, we talk about language equivalence of the resulting automaton to the original one (as in the above proposition). We have defined our automata without initial states, so we need to fix our notion of language equivalence of two automata.

▶ **Definition 5.** *Two DPAs $\mathcal{A}_1$ and $\mathcal{A}_2$ are called* language equivalent *if for each state p in one of the DPAs, there is a state q in the other DPA such that from p and q the same language is accepted (in the respective DPA).*

2.2 Schewe Merge

The main focus of this paper lies on techniques to generate merger templates such that a representative merge produces a language equivalent DPA. However, in the remainder of this section, we want to discuss a more involved merge operation than the representative merge. This operation is based on [17], and we therefore call it Schewe merge. We do not use this merge operation in the other sections. The aim is rather to illustrate that representative merges are not the only option.

The Schewe merge works rather similar to the representative merge. In addition to merging states from the merge sets into the chosen representative, it also redirects some transitions to the candidate set. While this does not remove additional states on its own, it simplifies the structure of the automaton to potentially improve the reduction of further reduction algorithms that are applied after the Schewe merge.

▶ **Definition 6.** *Let μ be a merger template. A* Schewe merge *of a DPA $\mathcal{A}$ w.r.t. μ is constructed by first building a representative merge. Then, for all merge sets M in μ and all transitions $\delta(p, a) = q$ in the original automaton, if $q \in \mu(M)$ and p is not reachable from q, then the transition is redirected to r_M instead.*

A Schewe merge differs from the representative merge if there is more than one state from the candidate set remaining, and the states are distributed over multiple SCCs. Whenever a transition would move the automaton to a candidate while changing SCC at the same time, that transition is instead redirected to the chosen representative state. One can imagine that, for example, this potentially enhances the reduction of a consecutive Moore merger, as more states now uniformly target the same representative.

It is not obvious if one can simply replace the representative merge with the Schewe merge and still keep the same properties such as preservation of language. We can identify a set of requirements that merger templates have to satisfy to be compatible with the Schewe merge.

▶ **Definition 7.** *For a representative merge $\mathcal{A}'$ of $\mathcal{A}$ w.r.t. μ, we define the candidate relation $\sim_C^\mu$ over the states of $\mathcal{A}'$ by $p \sim_C^\mu q$ if and only if $p = q$ or there is a $C \in \mu(D)$ with $p, q \in C$.*

We call μ 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 equivalence class of $\sim_C^\mu$ is symmetric (that is, if one state is reachable from another, they are in the same SCC).*

An example for a Schewe suitable merger template is $\mu_{\mathrm{LSF}}^{-1,\equiv_L}$ from Section 6. This merger template intuitively expresses that for each class of language equivalent states, one only needs to keep those in a "latest" SCC. The Schewe merge then corresponds to Construction 12 of [17] (where in [17] a notion of "almost equivalence" is used instead of language equivalence because the operation is used in the context of automata on finite words).

▶ **Theorem 8.** *Let μ be a Schewe suitable merger template and let $\mathcal{A}$ be a DPA. If a representative merge and a Schewe merge of $\mathcal{A}$ are built with the same choices for the representative states, then these two merge DPAs are language equivalent.*

Proof. Let $\mathcal{A}'$ be the representative merge and $\mathcal{A}''$ be the Schewe merge. Let q_0 be some starting state for the three runs ρ, ρ', ρ'' of the three automata on some word α. We claim that ρ' and ρ'' have the same acceptance status.

Let K be the set of positions where ρ'' uses a transition that does not exist in ρ'. We can observe that for every equivalence class κ of $\sim^{\mu}_{\mathcal{C}}$, there is at most one k_κ in K. If there would be two such positions k_κ and l_κ, then $\rho''(l_\kappa - 1)$ would be reachable from $\rho''(k_\kappa)$ which contradicts the requirement for the redirection of that edge in the Schewe merge.

As $K = \{k_1, \ldots, k_n\}$ is finite, ρ'' eventually only uses transitions that are also present in ρ'. By induction on i, we can show that $\rho'(k_i + 1) \sim^{\mu}_{\mathcal{C}} \rho''(k_i + 1)$, in particular for $i = n$. As $\sim^{\mu}_{\mathcal{C}}$ implies language equivalence by assumption, that means ρ' and ρ'' must have the same acceptance status. ◀

3 Delayed Simulation

We adapt the notion of delayed simulation, which has been introduced for alternating parity automata in [8], to DPAs. In the special case of DPAs, the computation of delayed simulation becomes simpler, and it can directly be used for state space reduction, while alternating and nondeterministic automata require more restricted variants for this purpose [8].

▶ **Definition 9** (adapted from [8]). *The* delayed simulation equivalence relation *is defined as* $p \equiv_{de} q$ *if and only if the following property holds for all* $w \in \Sigma^*$*: Let* $p' = \delta^*(p, w)$ *and* $q' = \delta^*(q, w)$*. Every run in the automaton that starts in* p' *or* q' *eventually sees a priority less than or equal to* $\min\{c(p'), c(q')\}$*.*

It is easy to see that $\equiv_{\text{de}}$ is a congruence relation that implies language equivalence. However, states that are $\equiv_{\text{de}}$-equivalent do in general not have the same priority. In order to correctly merge $\equiv_{\text{de}}$-equivalent states, one has to pick a representative of minimal priority from each class.

▶ **Definition 10.** *The* delayed simulation merge template *is* $\mu_{de} : \mathfrak{C}(\equiv_{de}) \to 2^Q$ *with* $\mu_{de}(\kappa) = \{q \in \kappa \mid c(q) = \min c(\kappa)\}$*.*

▶ **Theorem 11.** *A representative merge of a DPA* $\mathcal{A}$ *w.r.t.* μ_{de} *is language equivalent to the original.*

Proof. Consider an input word α, a run $\rho = q_0 q_1 \cdots$ of $\mathcal{A}$ on α from some state q_0, and the corresponding run ρ' starting in the $\equiv_{\text{de}}$-class of q_0 of the DPA $\mathcal{A}'$ obtained by a representative merge. Since the merge picks from each class a representative with smallest priority, it is clear that the priorities of states in ρ' are at each position smaller than or equal to the priorities in ρ. If ρ' visits a state of priority k, then ρ visits a state of priority k now or later, by definition of delayed simulation. Hence, the smallest priority that occurs infinitely often is the same in both runs. ◀

In [8] it is shown that delayed simulation can be computed by solving a Büchi game. Since we consider the special case of deterministic automata, we instead obtain just a deterministic Büchi automaton for which one has to solve language universality in order to compute the delayed simulation equivalence. The automaton is obtained by a product construction for tracking two runs, and a third component that keeps track of the smallest priority that the second state still has to match (see Lemma 13 below).

▶ **Definition 12.** *Define the deterministic Büchi automaton* $\mathcal{G}_{de} = (Q_{de}, \Sigma, \delta_{de}, F_{de})$ *as*

- $Q_{de} = Q \times Q \times (c(Q) \cup \{\checkmark\})$
- $\delta_{de}((p, q, k), a) = (p', q', \gamma(c(p'), c(q'), k))$*, where* $p' = \delta(p, a)$ *and* $q' = \delta(q, a)$

with obligation function

$$\gamma(i,j,k) = \begin{cases} \checkmark & \text{if } j \leq i \text{ and } j \leq_\checkmark k \\ \min_{\leq_\checkmark}\{i,k\} & \text{otherwise} \end{cases}$$

where $0 \leq_\checkmark 1 \leq_\checkmark 2 \leq_\checkmark \cdots \leq_\checkmark \checkmark$.

- $F_{de} = Q \times Q \times \{\checkmark\}$.

Now using this automaton, we can relate delayed simulation to the question of universal language. A state is *language universal* if starting from it, every input word is accepted.

▶ **Lemma 13.** *For two states p and q, let $q^0_{de}(p,q) = (p,q,\gamma(c(p),c(q),\checkmark))$. Then $p \equiv_{de} q$ if and only if $q^0_{de}(p,q)$ and $q^0_{de}(q,p)$ are language universal states (all infinite words are accepted from these states in $\mathcal{G}_{de}$).*

Proof. The run from $q^0_{\mathrm{de}}(p,q)$ in $\mathcal{G}_{\mathrm{de}}$ consists of the two runs from p and q in the original DPA $\mathcal{A}$, and the "obligations" in the third component. This obligation is the smallest number k such that the run from p has seen priority k, and the run from q has since then not seen a priority $\leq k$. The obligation is $\checkmark$ if no such number k exists. If the obligation becomes $\checkmark$ infinitely often, then for all priorities k seen in the run from p at some position i, the run from q visits a priority $\leq k$ at position i or later. With this observation, it follows that the condition from the lemma captures the definition of delayed simulation. ◀

▶ **Theorem 14.** *μ_{de} can be computed in $\mathcal{O}(|Q|^2 \cdot |c(Q)|)$.*

Proof. Assuming that we can compute $\equiv_{\mathrm{de}}$ in a suitable data structure in the described time, building μ_{de} from that is rather trivial. To see how we compute $\equiv_{\mathrm{de}}$, observe that the size of $\mathcal{G}_{\mathrm{de}}$ is $\mathcal{O}(|Q|^2 \cdot |c(Q)|)$. The set of language universal states in a DBA can be computed in linear time: we are looking for loops in the subgraph that only consists of the non-accepting states. Then, every state from which such a loop is reachable is not language universal. These operations can all be done in linear time with classic graph algorithms such as depth first search. ◀

4 Path Refinement

In this section, we present our first new technique, which we call path refinement. It starts from a given congruence relation $\sim$ on the state space that implies language equivalence. For path refinement, we pick one congruence class λ of $\sim$. We then define an equivalence relation only on the states of λ, and merge the states in the corresponding equivalence classes. For defining the equivalence relation of path refinement, we consider the set $L_{\lambda\hookleftarrow}$ of non-empty finite words that, starting in a state in λ lead the DPA back to λ without an intermediate visit to λ (note that the precise starting state inside λ does not matter because $\sim$ is a congruence):

$$L_{\lambda\hookleftarrow} := \{w = a_1 \cdots a_n \in \Sigma^+ \mid \begin{array}{l} \text{for all (or equivalently some) } q \in \lambda\text{: } \delta^*(q,w) \in \lambda, \text{ and} \\ \delta^*(q, a_1 \cdots a_i) \notin \lambda \text{ for all } 1 \leq i < n\}. \end{array}$$

Based on this language, we define the equivalence relation as follows.

▶ **Definition 15.** *Let $\sim$ be a congruence relation that implies language equivalence, and let $\lambda \in \mathfrak{C}(\sim)$ be an equivalence class. We define a relation R_λ on λ as $(p,q) \in R_\lambda$ if and only if for all $w \in L_{\lambda\hookleftarrow}$, the smallest priority seen on the path induced by w is the same starting from p and from q.*

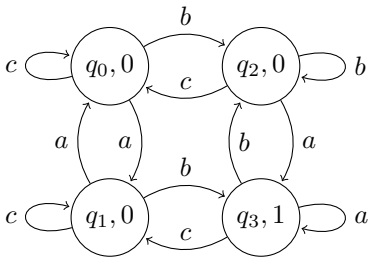

Figure 1 Example automaton.

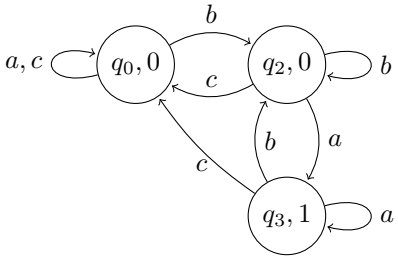

Figure 2 Example automaton after merging with $\mu_{\mathrm{PR}}^{\{q_0,q_1\}}$.

We define path refinement equivalence $\equiv_{PR}^{\lambda}$ *on* λ *as the largest subset of* R_λ *such that* $p \equiv_{PR}^{\lambda} q$ *if and only if for all* $w \in L_{\lambda\hookleftarrow}$, $\delta^*(p,w) \equiv_{PR}^{\lambda} \delta^*(q,w)$.

As an example, consider the DPA shown in Figure 1. For the relation $\sim$, we use exact language equivalence between states. In this case, the language equivalent states are $\{q_0, q_1\}$ and $\{q_2, q_3\}$. They are separated, e.g., by the word a^ω. We choose $\lambda = \{q_0, q_1\}$.

The set $L_{\lambda\hookleftarrow}$ is described by the regular expression $a + c + b(a + b)^*c$; reading any of these words from either q_0 or q_1 will take the DPA back to λ again.

Since q_0, q_1 both have the lowest priority 0, on every path for a word in $L_{\lambda\hookleftarrow}$, the lowest priority that is seen is 0. Hence, $q_0 \equiv_{\mathrm{PR}}^{\lambda} q_1$.

As for delayed simulation, the corresponding merger template defines for each class the states with smallest priority as candidates:

▶ **Definition 16.** *The* path refinement merger template *is* $\mu_{PR}^{\lambda} : \mathfrak{C}(\equiv_{PR}^{\lambda}) \to 2^Q$ *with* $\mu_{PR}^{\lambda}(\kappa) = \{q \in \kappa \mid c(q) = \min c(\kappa)\}$.

Going back to the example, the merger template would assign $\mu_{\mathrm{PR}}^{\lambda}(\{q_0, q_1\}) = \{q_0, q_1\}$. The representative merge for the candidate q_0 is shown in Figure 2.

Path refinement thus is able to remove one state from the automaton. In contrast, no two different states are in the delayed simulation equivalence relation.

One can check that this automaton is equivalent to the original DPA. The fact that this is true in general, is captured by the following theorem.

▶ **Theorem 17.** *A representative merge of a DPA* $\mathcal{A}$ *w.r.t.* μ_{PR}^{λ} *is language equivalent to the original.*

Proof. Let $\mathcal{A}'$ be the representative merge. Assume there is a starting state $q_0 \in Q'$ and a word α such that the acceptance of the runs ρ (of $\mathcal{A}$ starting in q_0) and ρ' (of $\mathcal{A}'$ starting in q_0) differs. We will bring this assumption to a contradiction.

First, note that at every position i, $\rho(i)$ and $\rho'(i)$ must be $\sim$-equivalent, as $\sim$ is a congruence relation. If in these runs, λ is visited only finitely often, there is a position j at which it is visited for the last time. Then from j on, ρ' only uses transitions that also exist in the original DPA $\mathcal{A}$. As $\rho(j) \sim \rho'(j)$, they must be language equivalent and therefore have the same acceptance status. This contradicts the assumption.

Otherwise, λ is visited infinitely often. However, for two consecutive positions k and k' at which λ is seen, we can show that the smallest priorities in $c(\rho(k)), \ldots, c(\rho(k'))$ and $c'(\rho'(k)), \ldots, c'(\rho'(k'))$ are the same. Then, it easily follows that the entire runs share the same smallest priority that is seen infinitely often.

To observe that the two run segments see the same minimal priority, first observe that $\rho(k) \equiv_{\mathrm{PR}}^{\lambda} \rho'(k)$ by induction on k. If k is the first position at which λ is visited, then $\rho'(k)$

is the representative of the equivalence class of $\rho(k)$ and therefore $\equiv^{\lambda}_{\mathrm{PR}}$-equivalent to $\rho(k)$. Then, by definition of the path refinement equivalence, the same holds for k' and therefore all following positions.

Now that we have established $\rho(k) \equiv^{\lambda}_{\mathrm{PR}} \rho'(k)$, it follows directly from the definition of R_λ that the smallest priorities in $c(\rho(k)), \dots, c(\rho(k'))$ and $c'(\rho'(k)), \dots, c'(\rho'(k'))$ are equal. ◀

We now turn to the question how to compute $\equiv^{\lambda}_{\mathrm{PR}}$ efficiently. In the naive approach, one can build a product automaton similar to the one for delayed simulation, in which the third component tracks the smallest priority so far and the component it was seen in. Then, at every visit to λ, the tracked values need to coincide. An algorithm based on such a product would have a complexity that is at least quadratic in the state space.

Instead, we build a Moore automaton of size $|Q| \cdot |c(Q)|$ that tracks only for single states the smallest priority seen on paths from λ back to λ. Moore equivalence in this automaton then corresponds to $\equiv^{\lambda}_{\mathrm{PR}}$.

▶ **Definition 18.** *Define the Moore automaton* $\mathcal{A}_{visit} = (Q^{\lambda}_{visit}, \Sigma, \delta^{\lambda}_{visit}, f^{\lambda}_{visit})$ *by*

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

▶ **Lemma 19.** *For a state* $q \in Q$*, let* $\iota_q = (q, \max c(Q)) \in Q^{\lambda}_{visit}$*. Then, for all states* p *and* q*, it holds that* $p \equiv^{\lambda}_{PR} q$ *if and only if* $\iota_p \equiv_M \iota_q$*.*

Proof. Our first observation is that for any state $p \in \lambda$, reading some $w \in L_{\lambda\hookleftarrow}$ from (p,k) ends in (q,k'), where k' is the smallest priority that occurs on the run segment.

If $p \not\equiv^{\lambda}_{\mathrm{PR}} q$, then there is a $w \in L_{\lambda\hookleftarrow}$ such that either the smallest priority when reading w from p and q differs, or reading w moves to non-PR-equivalent states. If the former is true, then reading w from ι_p and ι_q brings the visit graph to states with different priorities and therefore $\iota_p \not\equiv_M \iota_q$. If the former is false and the latter is true, then one has to repeatedly apply this argument until at some point a state pair is reached at which the first case is violated. This must happen eventually, as $\equiv^{\lambda}_{\mathrm{PR}}$ is defined as the *largest* subset satisfying its conditions.

For the other direction, if $\iota_p \not\equiv_M \iota_q$, there must be a $w \in \Sigma^*$ such that the priority differs when reading w from ι_p and ι_q. As all states not in λ have the same output $\bot$, we can split $w = v_1 \dots v_n$ such that all v_i are words in $L_{\lambda\hookleftarrow}$. Then, on the last segment, reading v_n sees different minimal priorities from the initial states, and therefore p and q cannot be PR-equivalent. ◀

▶ **Theorem 20.** $\equiv^{\lambda}_{PR}$ *can be computed in* $\mathcal{O}(|Q| \cdot |c(Q)| \cdot \log |Q|)$*.*

Proof. Moore equivalence for automata with n states can be computed in time $\mathcal{O}(n \log n)$ [9]. The number of states of $\mathcal{A}_{\mathrm{visit}}$ is in $\mathcal{O}(|Q| \cdot |c(Q)|)$. As $|c(Q)|$ is always at most $|Q|$, this gives us the desired complexity. ◀

5 Threshold Moore

Similar to Section 4, we again start from an equivalence relation $\sim$ on the state set Q of the DPA that implies language equivalence. In this section, $\sim$ does not have to be a congruence relation. We then intersect $\sim$ with a weakened version of Moore equivalence, and show that states that are equivalent in this intersection can be merged.

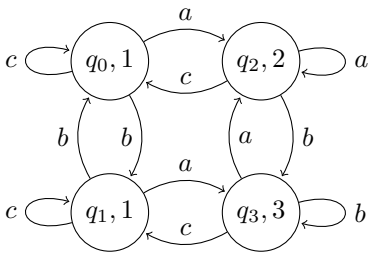

Figure 3 Example automaton.

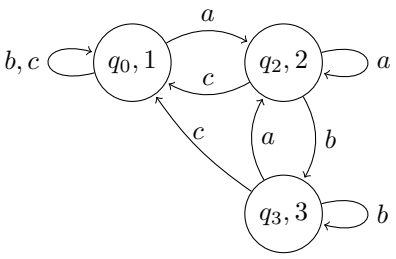

Figure 4 Example automaton after merging with $\mu^{\sim}_{\mathrm{TM}}$.

▶ **Definition 21.** *For a priority k, we define the* threshold Moore equivalence relation *as $p \equiv^{\leq k}_{M} q$ if and only if for all finite words w, $\delta^*(p,w)$ and $\delta^*(q,w)$ have the same priority or both priorities are greater than k.*

Let $\sim$ be an equivalence relation that implies language equivalence. We define the TM equivalence relation *as $p \equiv^{\sim}_{TM} q$ if and only if $p \sim q$, $c(p) = c(q)$, and $p \equiv^{\leq c(p)}_{M} q$.*

Note that for each k, the relation $\equiv^{\leq k}_{M}$ is a congruence but, in general, $\equiv^{\sim}_{\mathrm{TM}}$ is not a congruence, even if $\sim$ is (as can be seen in the example below).

Figure 3 shows a DPA on which we want to illustrate the reduction process. For $\sim$, we use exact language equivalence again. In this example, all four states are equivalent, as all accept the language $(a+b+c)^*(b^*a)^\omega$.

The threshold Moore relation depends on the choice for parameter k. For $k = 0$, all four states are equivalent because all states have priority greater than 0. For $k = 1$, there are three equivalence classes, $\{q_0, q_1\}$, $\{q_2\}$, and $\{q_3\}$. For $k > 1$, the relation becomes the same as $\equiv_M$ and all states are separated. These observations together imply that $q_0 \equiv^{\sim}_{\mathrm{TM}} q_1$, and these are the only states that are equivalent w.r.t. $\equiv^{\sim}_{\mathrm{TM}}$. Therefore, $\equiv^{\sim}_{\mathrm{TM}}$ is not a congruence because, for example, $\delta(q_0, a) = q_2$ and $\delta(q_1, a) = q_3$.

The merger template for TM relation simply merges classes of $\equiv^{\sim}_{\mathrm{TM}}$. Note that this is not, however, a quotient automaton, as $\equiv^{\sim}_{\mathrm{TM}}$ is in general not a congruence relation.

▶ **Definition 22.** *We define the* TM merger template $\mu^{\sim}_{TM} : \mathfrak{C}(\equiv^{\sim}_{TM}) \to 2^Q$ *as* $\mu^{\sim}_{TM}(\kappa) = \kappa$.

Continuing the example, the representative merge with the candidate q_0 for the class $\{q_0, q_1\}$, results in the automaton shown in Figure 4.

No distinct states are delayed simulation equivalent in this example. Furthermore, for the only $\sim$-class $\lambda = \{q_0, q_1, q_2, q_3\}$, one can check that no two distinct states are $\equiv^{\lambda}_{\mathrm{PR}}$-equivalent.

▶ **Lemma 23.** *Let $\mathcal{A}$ be a DPA and let $\mathcal{A}'$ be a representative merge w.r.t. a single equivalence class $\kappa \in \mathfrak{C}(\mu^{\sim}_{TM})$. Then $L(\mathcal{A}, q) = L(\mathcal{A}', q)$ for all states q of $\mathcal{A}'$. Furthermore, if k is the priority of the states in κ, then for all states p, q of $\mathcal{A}'$ with $k \geq c(p), c(q)$, we have $p \equiv^{\leq k}_{M} q$ in $\mathcal{A}$ if, and only if, $p \equiv^{\leq k}_{M} q$ in $\mathcal{A}'$.*

Proof. We focus on the language equivalence first. Let ρ and ρ' be the runs of the two automata on some word α starting in q. We show that these two runs have the same acceptance status.

Note that all states in ρ' are also states of $\mathcal{A}$. Since $\equiv_L$ is a congruence relation and only language equivalent states are merged, we have that $\rho(i) \equiv_L \rho'(i)$ in $\mathcal{A}$ for all positions i. The same is true for $\equiv^{\leq k}_{M}$.

If ρ visits infinitely many states of priority at most k, then the two runs see the same smallest priority $l < k$ infinitely often, as $c(\rho(i)) = l$ if and only if $c'(\rho'(i)) = l$. Thus, they must have the same acceptance status.

If $c(\rho)$ only visits finitely many states of priority at most k, then from some point j on in ρ', only transitions that also exist in $\mathcal{A}$ are taken. As $\rho(j) \equiv_L \rho'(j)$ in $\mathcal{A}$, we obtain that the two runs have the same acceptance status.

Regarding the second claim of the lemma, let p, q be states with $k \geq c(p), c(q)$ such that p, q are $\equiv_M^{\leq k}$-equivalent in $\mathcal{A}$. Let $\alpha \in \Sigma^\omega$, and consider the runs ρ, π of $\mathcal{A}$ on α from p, q, as well as the run π', ρ' of $\mathcal{A}'$ on α starting in p, q.

As $\equiv_M^{\leq k}$ is a congruence relation, $\rho(i) \equiv_M^{\leq k} \rho'(i)$ and $\pi(i) \equiv_M^{\leq k} \pi'(i)$ in $\mathcal{A}$ for all positions i. Furthermore, since p and q are $\equiv_M^{\leq k}$-equivalent in $\mathcal{A}$, also $\rho(i) \equiv_M^{\leq k} \pi(i)$ in $\mathcal{A}$ for all i. This implies that $\rho'(i) \equiv_M^{\leq k} \pi'(i)$ in $\mathcal{A}$ for all i.

Therefore, at the positions at which one of ρ' and π' visits a priority $\leq k$, the other run visits the same priority. Hence, p, q are $\equiv_M^{\leq k}$-equivalent in $\mathcal{A}'$. ◀

▶ **Theorem 24.** *A representative merge of a DPA w.r.t. $\mu_{TM}^{\sim}$ is language equivalent to the original.*

Proof. Let $\kappa_1, \ldots, \kappa_m$ be an enumeration of the equivalence classes in $\mu_{\mathrm{TM}}^{\sim}$ sorted by descending priority. By Lemma 23, merging the states in κ_i will not change the equivalence classes $\kappa_{i+1}, \ldots, \kappa_n$. It is therefore a language preserving operation to merge all equivalence classes in the given order. The resulting automaton is the same as a representative merge w.r.t. $\mu_{\mathrm{TM}}^{\sim}$. ◀

The computation of $\mu_{\mathrm{TM}}^{\sim}$ is rather straightforward.

▶ **Theorem 25.** *For a given $\sim$ in a suitable data structure, $\mu_{TM}^{\sim}$ can be computed in time $\mathcal{O}(|Q| \cdot |c(Q)| \cdot \log |Q|)$.*

Proof. Assuming that $\equiv_{\mathrm{TM}}^{\sim}$ is known, computing $\mu_{\mathrm{TM}}^{\sim}$ is easy. For obtaining $\equiv_{\mathrm{TM}}^{\sim}$, one needs the relations $\equiv_M^{\leq k}$. For each k, this can be computed with just a slight adaption of usual algorithms for Moore equivalence in time $\mathcal{O}(|Q| \cdot \log |Q|)$. This needs to be done for every k, so $|c(Q)|$ times. ◀

6 Labeled SCC Filter

The labeled SCC filter technique (LSF) is also based on the threshold Moore equivalence from Definition 21. While in Section 5 only states of priority k could be merged based on $\equiv_M^{\leq k}$, we now consider states that are $\equiv_M^{\leq k}$-equivalent, have priority greater than k, and are in different SCCs after removing all states with priority $\leq k$. We then keep from each equivalence class only those states that are in a "deepest" SCC in this restricted automaton, in the sense that no other SCCs are reachable from it.

For that purpose, let $\mathcal{A} \restriction_{>k}^{c}$ be the restriction of $\mathcal{A}$ to states with priority greater than k. Furthermore, we let $\preceq_k$ be a total preorder on the states in $\mathcal{A} \restriction_{>k}^{c}$ that extends the reachability preorder. More formally, if q is reachable from p in $\mathcal{A} \restriction_{>k}^{c}$, then $p \preceq_k q$; on the other hand, if q is not reachable from p, then either $p \prec_k q$ or $q \prec_k p$.

In other words, $\preceq_k$ is a preorder whose equivalence classes are exactly the SCCs in $\mathcal{A} \restriction_{>k}^{c}$ and which is compatible with a topological sorting of the states.

▶ **Definition 26.** *Let $k \geq -1$ and let $\sim$ be an equivalence relation that implies language equivalence. We define the* LSF equivalence relation $\equiv_{LSF}^{k,\sim}$ *such that two states p and q are equivalent if and only if*

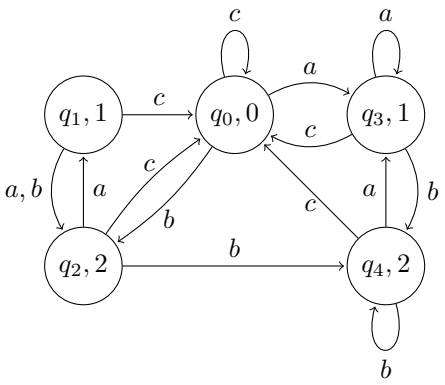

Figure 5 Example automaton.

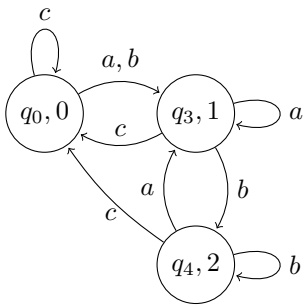

Figure 6 Example automaton after merging with $\mu^{0,\sim}_{\mathrm{LSF}}$.

- $p = q$; *or*
- $c(p) > k$, $c(q) > k$, $p \equiv^{\leq k}_{M} q$, *and* $p \sim q$.

Consider the DPA shown in Figure 5. As $\sim$, we use language equivalence as in the previous examples. All five states are language equivalent. We choose $k = 0$. Since $\sim$ has only one class, $\equiv^{0,\sim}_{\mathrm{LSF}}$ is the same as the threshold Moore relation for $k = 0$, which consists of the two equivalence classes $\{q_0\}$ and $\{q_1, q_2, q_3, q_4\}$, separated by the empty word.

The LSF merger template selects for each equivalence class the maximal elements w.r.t. $\preceq_k$ as candidates. Formally, we also need to treat the states with priority $\leq k$, each of which forms its own singleton equivalence class.

▶ **Definition 27.** *For each equivalence class κ of $\equiv^{k,\sim}_{LSF}$: if $\kappa = \{q\}$ for $c(q) \leq k$, then $M^k_\kappa = C^k_\kappa = \{q\}$, and otherwise let*

$$C^k_\kappa = \{r \in \kappa \mid p \preceq_k r \text{ for all } p \in \kappa\} \text{ and } M^k_\kappa = \kappa \setminus C^k_\kappa.$$

Define the LSF merger template *by $\mu^{k,\sim}_{LSF}(M^k_\kappa) = C^k_\kappa$ for each equivalence class κ of $\equiv^{k,\sim}_{LSF}$.*

We continue our example from before with the class $\kappa = \{q_1, q_2, q_3, q_4\}$. Keeping only the states with priority greater than 0, i.e. removing q_0 from the automaton, breaks it into the two SCCs $\{q_1, q_2\}$ and $\{q_3, q_4\}$. There is a transition from q_2 to q_4, so the relation $\preceq_0$ is given by $\{q_1, q_2\} \prec_0 \{q_3, q_4\}$.

The merger template therefore assigns $\mu^{0,\sim}_{\mathrm{LSF}}(\{q_1, q_2\}) = \{q_3, q_4\}$. Deciding on q_3 as the representative, the resulting automaton after the merge is displayed in Figure 6.

None of the previous reduction algorithms, that is, delayed simulation, path refinement based on $\sim$, or threshold Moore are able to remove a state from the original automaton.

▶ **Lemma 28.** *Let $\mathcal{A}$ be a DPA and let $\mathcal{A}'$ be a representative merge w.r.t. a single equivalence class $\kappa \in \mathfrak{C}(\mu^{k,\sim}_{LSF})$. Then, $L(\mathcal{A}, q) = L(\mathcal{A}', q)$ for all states q of $\mathcal{A}'$. Furthermore, for all states p, q of $\mathcal{A}'$, we have $p \equiv^{k,\sim}_{LSF} q$ in $\mathcal{A}$ if, and only if, $p \equiv^{k,\sim}_{LSF} q$ in $\mathcal{A}'$.*

Proof. Let ρ and ρ' be the runs of the two automata on some word α starting in q. We show that these two runs have the same acceptance status. Since, by definition, $\equiv^{k,\sim}_{\mathrm{LSF}} \subseteq \equiv_L$, and $\equiv^{k,\sim}_{\mathrm{LSF}} \subseteq \equiv^{\leq k}_{M}$, we have that $\rho(i) \equiv_L \rho'(i)$ and $\rho(i) \equiv^{\leq k}_{M} \rho'(i)$ in $\mathcal{A}$ for all positions i (because $\equiv_L$ and $\equiv^{\leq k}_{M}$ are congruences). The definition of $\equiv^{\leq k}_{M}$ implies that both runs visit priorities $\leq k$ at the same positions. If infinitely many such priorities $\leq k$ are visited, this implies that both runs have the same acceptance status. If finitely many priorities $\leq k$ are visited,

note that between two transitions in $\mathcal{A}'$ that are not in $\mathcal{A}$, there has to be a priority $\leq k$ by definition of the merger template. Hence, in this case, ρ' uses only finitely many transitions that are not in $\mathcal{A}$. Let j be some position such that ρ' after j only uses transitions that also exist in $\mathcal{A}$. Since $\rho(j) \equiv_L \rho'(j)$, as noted earlier, we obtain that ρ and ρ' have the same acceptance status.

Concerning the second statement, let p, q be states of $\mathcal{A}'$ with $p \equiv^{k,\sim}_{\mathrm{LSF}} q$ in $\mathcal{A}$. We show that $p \equiv^{k,\sim}_{\mathrm{LSF}} q$ in $\mathcal{A}'$ by proving $p \equiv^{\leq k}_{M} q$ and $p \equiv_L q$ in $\mathcal{A}'$. Note that $\sim$ in $\mathcal{A}'$ is just the restriction of $\sim$ to the state set of $\mathcal{A}'$, so showing $p \equiv_L q$ in $\mathcal{A}'$ also implies that $p \sim q$.

Let α be an infinite word, and let π, π' be the runs of $\mathcal{A}, \mathcal{A}'$ on α starting in p, and ρ, ρ' be the runs of $\mathcal{A}, \mathcal{A}'$ on α starting in q. As explained above, we have $\pi(i) \equiv^{\leq k}_{M} \pi'(i)$ and $\rho(i) \equiv^{\leq k}_{M} \rho'(i)$ in $\mathcal{A}$ for all i. Furthermore, $p \equiv^{k,\sim}_{\mathrm{LSF}} q$ in $\mathcal{A}$ implies that also $\pi(i) \equiv^{\leq k}_{M} \rho(i)$ in $\mathcal{A}$ for all i. By transitivity, we obtain that $\pi'(i) \equiv^{\leq k}_{M} \rho'(i)$ in $\mathcal{A}$. In particular, π' and ρ' are at the same time in states of priority $\leq k$. Since α was picked arbitrarily, we conclude that $p \equiv^{\leq k}_{M} q$ in $\mathcal{A}'$.

For showing $p \equiv_L q$ in $\mathcal{A}'$, note that we have shown above (for the first claim of the lemma) that from p the same words are accepted in $\mathcal{A}$ and $\mathcal{A}'$, and from q the same words are accepted in $\mathcal{A}$ and $\mathcal{A}'$. Since $p \equiv_L q$ in $\mathcal{A}$, we can conclude that also $p \equiv_L q$ in $\mathcal{A}'$. ◀

▶ **Theorem 29.** *A representative merge of a DPA w.r.t. $\mu^{k,\sim}_{LSF}$ is language equivalent to the original.*

Proof. Let $\kappa_1, \ldots, \kappa_m$ be an enumeration of the equivalence classes of $\equiv^{k,\sim}_{\mathrm{LSF}}$. When merging $M^k_{\kappa_i}$ into $C^k_{\kappa_i}$, the language is preserved and the equivalence classes $\kappa_{i+1}, \ldots, \kappa_m$ do not change by Lemma 28. Also the candidate sets $C^k_{\kappa_j}$ themselves do not change, so we can safely merge all $M^k_{\kappa_i}$ into $C^k_{\kappa_i}$. This is the same operation as performed by the merger template. ◀

Computation of the LSF merger consists of computing the threshold Moore equivalence and a reachability analysis in the restricted graph.

▶ **Theorem 30.** *For a given $\sim$ in a suitable data structure, $\mu^{k,\sim}_{LSF}$ can be computed in time $\mathcal{O}(|Q| \cdot \log |Q|)$.*

Proof. $\sim$ is already given and $\equiv^{\leq k}_{M}$ can be computed in $\mathcal{O}(|Q| \cdot \log |Q|)$. Building $\equiv^{k,\sim}_{\mathrm{LSF}}$ is an easy linear time intersection operation.

The second step to building $\mu^{k,\sim}_{\mathrm{LSF}}$ is to compute C^k_κ for each κ. For that, it suffices to find the order $\preceq_k$ and then select all the maximal elements from each equivalence class. This order can be computed by a topological sorting on the SCCs of $\mathcal{A} \restriction^c_{>k}$ which one can construct in linear time. ◀

7 Experimental Data

All algorithms that have been presented in Sections 3–6 were also implemented by us in C++ in order to evaluate them on larger examples. The test data set consisted of roughly 100 automata that were constructed from LTL specifications of the Reactive Synthesis Competition (SYNTCOMP) [10].

The automata were constructed by translating the given LTL formulas into nondeterministic Büchi automata using the Spot tool ([4]), followed by a conversion to a DPA using nbautils ([14]). During the generation of these DPAs, several techniques for state reduction are already applied, such as minimizing the number of priorities according to [3], and then minimizing the DPA as Moore automata.

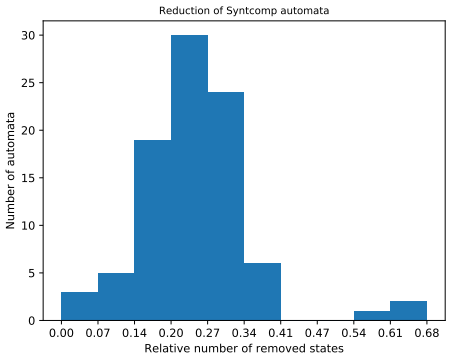

Figure 7 Reduction of SYNTCOMP automata. (all techniques)

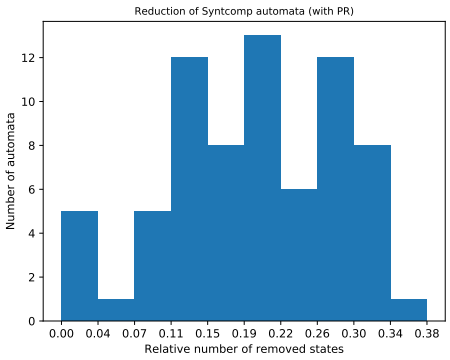

Figure 8 Reduction of SYNTCOMP automata. (only PR)

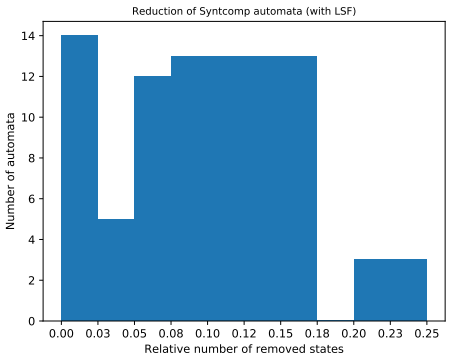

Figure 9 Reduction of SYNTCOMP automata. (only LSF)

The sizes of the testing automata range from 9 to 3575 states with a median of 48 and an average of 202; the size of the alphabet Σ ranges from 4 to 2048 symbols with a median of 16. There are three or four different priorities in most of the automata.

The techniques presented in Sections 4–6 require a given equivalence relation $\sim$ that implies language equivalence. Although language equivalence of states in DPAs can be computed in polynomial time, the space and time complexity of the algorithm is in $\mathcal{O}(|Q|^2|c(Q)|^2)$, which turns out to be too high for the larger examples. Instead, we use a relation $\sim$ that can be produced as a side-effect of the determinization construction, as explained in the following.

Determinization constructions for Büchi automata, like the Safra construction [16, 15], are refinements of the standard subset construction for NFAs. The set of states that could have been reached in the NBA is tracked in combination with additional information on visits to accepting states. So there are, in general, many states of the constructed DPA that correspond to the same set S of Büchi states. All these states in the DPA that correspond to the same set S are language equivalent, because from all of them precisely those words are accepted that are accepted by the Büchi automaton from one of the states in S. Therefore, the relation $\sim$ defined for states p, q of the DPA by $p \sim q$ iff p and q correspond to the same set S of Büchi states, is a congruence relation that implies language equivalence. It can therefore be used in the algorithms from Sections 4–6, and is obtained "for free" from the determinization construction.

Figure 7 shows a histogram of overall reduction that was achieved in our experiments. As the generated DPAs can reach sizes of more than 1000, the low complexity of the new techniques was very important. To obtain the histogram, all reduction techniques were applied in succession, with the exception of delayed simulation which proved to be too difficult for the largest automata with its quadratic complexity in both space and time.

The histogram shows a reduction between 14 and 34% of the states in most cases. Individually, the two approaches showing most reduction were path refinement (Section 4) and LSF (Section 6), both of which are analyzed in Figures 8 and 9.

Apart from these tests, we had our reduction also run on DPAs that were determinized from randomly constructed NBAs. The results are rather similar to those shown here and confirm our analysis. In addition to path refinement and LSF, also delayed simulation showed great potential on automata small enough. We consider automata from actual specifications to be of more relevance though, which is why we focus on the SYNTCOMP set here.

8 Conclusion

We have proposed three new ways of reducing the state space of DPAs, and analyzed the known technique of delayed simulation from [6, 8] in the context of DPAs. For obtaining a uniform way of describing the methods, we have introduced the notion of merger template, in order to capture different types of merge operations.

The equivalence relations on which our reduction techniques are based can all be computed very efficiently. Our experiments show that the new methods can further reduce the state space of DPAs that have been obtained by determinizing Büchi automata, and that have already been reduced with known techniques.

We therefore believe that the proposed methods provide interesting tools to be used as post-processing after determinization constructions that produce DPAs. Since other acceptance conditions, like Rabin or Streett automata, are also commonly used in algorithms, a possible topic for future research would be to see if and how our methods can be adapted to these conditions.

References

1 Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

2 J. Richard Büchi. On a decision method in restricted second order arithmetic. In *International Congress on Logic, Methodology and Philosophy of Science*, pages 1–11. Stanford University Press, 1962.

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

4 Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. Spot 2.0 — a framework for LTL and ω-automata manipulation. In *Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis (ATVA'16)*, volume 9938 of *Lecture Notes in Computer Science*, pages 122–129. Springer, October 2016. doi:10.1007/978-3-319-46520-3_8.

5 E. Allen Emerson and Chin-Laung Lei. Modalities for Model Checking: Branching Time Logic Strikes Back. *Sci. Comput. Program.*, 8(3):275–306, 1987. doi:10.1016/0167-6423(87)90036-0.

6 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.

7 Carsten Fritz and Thomas Wilke. Simulation Relations for Alternating Büchi Automata. *Theor. Comput. Sci.*, 338(1-3):275–314, June 2005. doi:10.1016/j.tcs.2005.01.016.

8 Carsten Fritz and Thomas Wilke. Simulation Relations for Alternating Parity Automata and Parity Games. In *Developments in Language Theory, 10th International Conference, DLT 2006, Santa Barbara, CA, USA, June 26-29, 2006, Proceedings*, pages 59–70, 2006. doi:10.1007/11779148_7.

9 John E. Hopcroft. An N Log N Algorithm for Minimizing States in a Finite Automaton. Technical report, Stanford University, Stanford, CA, USA, 1971.

10 Swen Jacobs, Guillermo A. Pérez, and Roderick Bloem. The Reactive Synthesis Competition. URL: http://www.syntcomp.org.

11 Richard Mayr and Lorenzo Clemente. Advanced Automata Minimization. In *POPL 2013*, October 2012. URL: http://arxiv.org/abs/1210.6624.

12 Philipp J. Meyer, Salomon Sickert, and Michael Luttenberger. Strix: Explicit Reactive Synthesis Strikes Back! In *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I*, pages 578–586, 2018. doi:10.1007/978-3-319-96145-3_31.

13 A.W. Mostowski. Hierarchies of weak automata and weak monadic formulas. *Theoretical Computer Science*, 83(2):323–335, 1991. doi:10.1016/0304-3975(91)90283-8.

14 Anton Pirogov. nbautils. https://github.com/apirogov/nbautils, 2018.

15 Nir Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. In *Logic in Computer Science, 2006 21st Annual IEEE Symposium on*, pages 255–264. IEEE, 2006.

16 Shmuel Safra. On the complexity of omega-automata. In *Foundations of Computer Science, 1988., 29th Annual Symposium on*, pages 319–327, 1988.

17 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. doi:10.4230/LIPIcs.FSTTCS.2010.400.

18 Fabio Somenzi and Roderick Bloem. Efficient Büchi Automata from LTL Formulae. In *Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings*, pages 248–263, 2000. doi:10.1007/10722167_21.

19 Wolfgang Thomas. Handbook of Formal Languages, Vol. 3. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages, Vol. 3*, chapter Languages, Automata, and Logic, pages 389–455. Springer-Verlag New York, Inc., New York, NY, USA, 1997. URL: http://dl.acm.org/citation.cfm?id=267871.267878.

20 Wolfgang Thomas. Church's Problem and a Tour through Automata Theory. In *Pillars of Computer Science, Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday*, pages 635–655. Springer, 2008. doi:10.1007/978-3-540-78127-1.

21 Moshe Y. Vardi and Thomas Wilke. Automata: from logics to algorithms. In *Logic and automata - history and perspectives*, volume 2 of *Texts in Logic and Games*, pages 629–724. Amsterdam University Press, 2007.