Rheinisch-Westfälische Technische Hochschule Aachen

Lehrstuhl für Informatik 7
Logik und Theorie diskreter Systeme

Master Thesis

State Space Reduction For Deterministic Parity Automata

by Andreas Tollkötter

Supervisor: Dr. Christof Löding

December 28, 2018

I hereby declare that I have created this work completely on my own and used no other sources or tools than the ones listed, and that I have marked any citations accordingly.

Aachen, December 28, 2018
Andreas Tollkötter

Abstract

Exact minimization of ω-automata is a difficult problem and heuristic algorithms are a subject of current research. We establish a framework to uniformly describe such algorithms and investigate several approaches to reduce the state space of deterministic parity automata. The description of these procedures consists of a theoretical analysis as well as data collected using an actual implementation.

Contents

Introduction

Finite automata are a long established computation model that dates back to sources such as [12] and [18]. 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 [9]. Regarding nondeterministic finite automata (NFA), [10] 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 [20] and spawned heuristic approaches such as [20], [11], or [6].

As [22] 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 ([14]), that use a parity function, assigning states to output numbers, rather than the usual acceptance set. A run of a parity automaton is accepting if the set of numbers that are output infinitely often has an even number as its minimum. [15], [23] 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 ([20]).

In this paper, we research heuristic state space reduction techniques for DPAs. As a first step, a general framework is introduced. These *merger functions* are a generalization of quotient automata which are e.g. used in the minimization of DFAs. The state set of an automaton is partitioned into multiple merge sets, each of which is mapped by the merger function to a set of candidate sets. A *representative merge* then chooses a representative candidate and removes all other states from the particular merge set, redirecting all transitions to the representative. As an alternative to the representative merge, we also describe an alternative approach called a *Schewe merge* based on ideas in [20].

The most basic merge simply adapts the algorithm from [9]. Every parity automaton can be interpreted as a Moore automaton which can then be minimized using said algorithm. 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*. That relation is used or modified at several later points of the thesis.

A simple merger function we introduce is that of the *skip merger* which takes an equivalence relation that implies language equivalence on the states as a parameter and from that builds a merger function. The idea of using one given equivalence relation on the states and refining it is used several times in the thesis. This merge decides on one particular strongly connected component (SCC) of the automaton and removes all states of an equivalence class that do not lie in this SCC, essentially "skipping" all other SCCs.

We adapt the works of [7], who worked on alternating automata, to our case of deterministic parity automata and find that there are enough differences to warrant a separate analysis. The *delayed simulation merger* considers two states to be equal, if on every run from those states on a shared word, if one run visits a priority at some position, the other run must visit a priority at most as high at some point in the future. It then holds that both runs will see the same smallest

priority infinitely often and therefore either both accept or both reject. It suffices to choose one representative of each such equivalence class that has minimal priority and build the quotient automaton with those representatives.

The *iterated Moore* merge uses the idea that on ω-automata, states which are only visited finitely often are irrelevant to the acceptance of a state. In particular, trivial SCCs, i.e. states with no path back to itself, can have their priority freely changed without affecting the language of the automaton. The idea of this merger therefore is to loosen the constraints on those states and then build the usual Moore equivalence.

We bring up merging via *path refinement*, which uses an existing equivalence class of some congruence relation and refines it to a point where states can safely be merged. This refinement occurs so that two states from the class are equivalent if on each path from that state back to the class, both states visit the same minimal priority.

Another merger function is called *threshold Moore*, which again refines an existing relation. Two states are considered to be equivalent under the refinement if a relaxation of Moore equivalence, that considers all priorities greater than some k to be equal, matches them. In this case we require the two states to have equal priority and choose the k to be that exact value.

A similar but slightly different approach is the *LSF* merger function. It removes the need for two states to have equal priority and instead takes the value k as a parameter. On the other side it adds a requirement similar to that of the skip merger, in that the candidates of the merge are those that all lie in one single SCC if we modify the automaton to only contain those states of priority at least k.

The thesis is structured as follows: In the first chapter, we define basics notations and conventions and present some data of our empirical testing environment used to collect practical data of each approach. In the second chapter, we establish some theoretical ground work that will be used by the rest of the thesis. After that, we present the algorithms for state space reduction, split into one chapter for each such procedure. Each such chapter is made up of at least three sections. The first section describes the idea and definition of the algorithm and proofs well behaving properties. The second section covers how to actually compute the reduction and provides a short run time analysis. The third section shows practical data to analyze "real world" usefulness. Potential other sections contain variants or extensions of the original procedure as well as potential open questions. At the very end, the appendix includes small examples to show how the individual merger functions are computed and work.

Chapter 1

Basics

1.1 Basic Definitions

1.1.1 Sets and Functions

Definition 1.1.1. The *natural numbers* $\mathbb{N} = \{0, 1, 2, \dots\}$ are the set of all non-negative integers. We define $0 := \emptyset$, $1 := \{0\}$, $2 := \{0, 1\}$, and so forth.

The value ω denotes the "smallest" infinity, $\omega := \mathbb{N}$. For all natural numbers, we write $n < \omega$ and $\omega \not< \omega$. Also, we sometimes use the convention $n + \omega = \omega$.

We denote the set $\mathbb{N} \cup \{\omega\}$ by $\mathbb{N}_\omega$.

Definition 1.1.2. Let X and Y be two sets. We use the usual definition of union ($\cup$), intersection ($\cap$), and set difference ($\setminus$). If some domain ($X \subseteq D$) is clear in the context, we write $X^\complement = D \setminus X$.

We use the Cartesian product $X \times Y = \{(x, y) \mid x \in X, y \in Y\}$.

We write X^Y for the set of all functions with domain Y and range X. If we have a function $f : D \to \{0, 1\}$, then we sometimes implicitly use it as a set $X \subseteq D$ with $x \in X$ iff $f(x) = 1$. In particular, 2^Y is the power set of Y.

Definition 1.1.3. Let $f : D \to R$ be a function and let $X \subseteq D$ and $Y \subseteq R$. We describe by $f(X) = \{f(x) \in R \mid x \in X\}$ and $f^{-1}(Y) = \{x \in D \mid \exists y \in Y : f(x) = y\}$.

Definition 1.1.4. Let $X \subseteq D$ be a set. For $D' \subseteq D$, we define $X \restriction_{D'} = X \cap D$. In particular, we use this notation for relations, e.g. $R \subseteq \mathbb{N} \times \mathbb{N}$ and $R \restriction_{\{0\} \times \mathbb{N}}$.

For a function $f : D \to R$, we write $f \restriction_{D'}$ for the function $f' : D' \to R, x \mapsto f(x)$.

1.1.2 Relations and Orders

Definition 1.1.5. Let X be a set. We call a set $R \subseteq X \times X$ a *relation* over X. R is

- *reflexive*, if for all $x \in X$, $(x, x) \in R$.
- *irreflexive*, if for all $x \in X$, $(x, x) \notin R$.
- *symmetric*, if for all $(x, y) \in R$, also $(y, x) \in R$.

- *asymmetric*, if for all $(x, y) \in R$, $(y, x) \notin R$.
- *transitive*, if for all $(x, y), (y, z) \in R$, also $(x, z) \in R$.
- *total*, if for all $x, y \in X$, $(x, y) \in R$ or $(y, x) \in R$ is true.

We call R

- a *partial order*, if it is irreflexive, asymmetric, and transitive.
- a *total order*, if it is a partial order and total.
- a *preorder*, if it is reflexive and transitive.
- a *total preorder*, if it is a preorder and total.
- an *equivalence relation*, if it is a preorder and symmetric.

If R is a partial order or a preorder, we call an element $x \in X$ *minimal* (w.r.t. R), if for all $y \in X$, $(y, x) \in R$ implies $(x, y) \in R$. Similarly, we call it *maximal*, if for all $y \in X$, $(x, y) \in R$ implies $(y, x) \in R$.

We call x the *minimum* of R if for all $y \neq x$, $(y, x) \in R$. We write $x = \min_R X$.

Definition 1.1.6. Let R be a partial order over X. We call a set $S \subseteq Y$ an *extension of R to Y* if $X \subseteq Y$, $R \subseteq S$, and S is a partial order over Y. We use the same notation for total orders, preorders, and total preorders.

Definition 1.1.7. Let R be an equivalence relation over X. R implicitly forms a partition of X into *equivalence classes*. For an element $x \in X$, we call $[x]_R := \{y \in X \mid (x, y) \in R\}$ the equivalence class of x. We denote the set of equivalence classes by $\mathfrak{C}(R) = \{[x]_R \mid x \in R\}$.

1.1.3 Words and Languages

Definition 1.1.8. A non-empty set of symbols can be called an *alphabet*, which we will denote by a variable Σ most of the time. As symbols, we usually use lower case letters, i.e. a or b.

A *finite word*, usually denoted by u, v, or w, over an alphabet Σ is a function $w : n \to \Sigma$ for some n. We call n the *length* of w and write $|w| = n$. The unique word of length 0 is called *empty word* and is written as ε.

Given $\Sigma^n = \{w \mid w \text{ is a word of length } n \text{ over } \Sigma\}$, we define $\Sigma^* = \bigcup_{n \in \mathbb{N}} \Sigma^n$ as the set of all finite words over Σ.

Definition 1.1.9. An *ω-word*, usually denoted by α or β, over an alphabet Σ is a function $\alpha : \omega \to \Sigma$. ω is the length of α and we write $|\alpha| = \omega$. The set Σ^ω then describes the set of all ω-words over Σ.

Definition 1.1.10. A *language* over an alphabet Σ is a set of words $L \subseteq \Sigma^* \cup \Sigma^\omega$. In the context we use it should always be clear whether we are using finite words or ω-words.

Definition 1.1.11. Let $v, w \in \Sigma^*$ and $w_i \in \Sigma^*$ for all $i \in \mathbb{N}$ be words over Σ and $\alpha \in \Sigma^\omega$ be an ω-word over Σ.

The *concatenation* of v and w (denoted by $v \cdot w$) is a word u such that:

$$u : |v| + |w| \to \Sigma, i \mapsto \begin{cases} v(i) & \text{if } i < |v| \\ w(i - |v|) & \text{else} \end{cases}$$

The *concatenation* of w and α (denoted by $w \cdot \alpha$) is an ω-word β such that:

$$\beta : \mathbb{N} \to \Sigma, i \mapsto \begin{cases} w(i) & \text{if } i < |w| \\ \alpha(i - |w|) & \text{else} \end{cases}$$

For some $n \in \mathbb{N}$, the *n-iteration* of w (denoted by w^n) is a word u such that:

$$u : |w|^n \to \Sigma, i \mapsto w(i \mod |w|)$$

The *ω-iteration* of w (denoted by w^ω) is an ω-word α such that:

$$\beta : \mathbb{N} \to \Sigma, i \mapsto w(i \mod |w|)$$

For the purpose of easier notation and readability, we write singular symbols as words, i.e. for an $a \in \Sigma$ we write a for the word $w_a : \{0\} \to \Sigma, i \mapsto a$.
We also abbreviate $v \cdot w$ to vw and $w \cdot \alpha$ to $w\alpha$. Further, we use $\alpha \cdot \varepsilon = \alpha$ for $\alpha \in \Sigma^\omega$.

Definition 1.1.12. Let $L, K \subseteq \Sigma^*$ be a language and $U \subseteq \Sigma^\omega$ be an ω-language.
The *concatenation* of L and K is $L \cdot K = \{u \in \Sigma^* \mid \text{There are } v \in L \text{ and } w \in K \text{ such that } u = v \cdot w\}$.
The *concatenation* of L and U is $L \cdot U = \{\alpha \in \Sigma^\omega \mid \text{There are } w \in L \text{ and } \beta \in U \text{ such that } \alpha = w \cdot \beta\}$.
For some $n \in \mathbb{N}$, the *n-iteration* of L is $L^n = \{w \in \Sigma^* \mid \text{There is } v \in L \text{ such that } w = v^n\}$.
The *Kleene closure* of L is $L^* = \bigcup_{n \in \mathbb{N}} L^n$.

Definition 1.1.13. Let $w \in \Sigma^* \cup \Sigma^\omega$ be a word. We define a sub string or sub word of w for some $n \leq m \leq |w|$ as $w[n, m] = w(n) \cdot w(n+1) \cdots w(m-1)$. In the case that $m = |w| = \omega$, it is simply $w[n, m] = w(n) \cdot w(n+1) \cdots$. Note that for $n = m$, we have $w[n, m] = \varepsilon$.

Definition 1.1.14. Let $v, w \in \Sigma^* \cup \Sigma^\omega$ be words. We call v

- a *prefix* of w, if there is an $n \in \mathbb{N}_\omega$ with $v = w[0, n]$.
- a *suffix* of w, if there is an $n \in \mathbb{N}_\omega$ with $v = w[n, |w|]$.
- an *infix* of w, if there are $n, m \in \mathbb{N}_\omega$ with $v = w[n, m]$.

Definition 1.1.15. The *occurrence set* of a word $w \in \Sigma^* \cup \Sigma^\omega$ is the set of symbols which occur at least once in w.

$$\mathrm{Occ}(w) = \{a \in \Sigma \mid \text{There is an } n \in |w| \text{ such that } w(n) = a.\}$$

The *infinity set* of a word $w \in \Sigma^\omega$ is the set of symbols which occur infinitely often in w.

$$\mathrm{Inf}(w) = \{a \in \Sigma \mid \text{For every } n \in \mathbb{N} \text{ there is a } m > n \text{ such that } w(m) = a.\}$$

Definition 1.1.16. Let $w \in \Sigma^* \cup \Sigma^\omega$ be a word and $f : \Sigma \to \Gamma$ be a funcction. We define $f(w) \in \Gamma^{|w|}$ as $(f(w))(n) = f(w(n))$.

1.1.4 Automata

Definition 1.1.17. Let Q be a set, Σ an alphabet, and $\delta : Q \times \Sigma \to Q$ a function. We call $\mathcal{S} = (Q, \Sigma, \delta)$ a *deterministic transition structure*. We call Q the states or state space.

For $q \in Q$ and a word $w \in \Sigma^* \cup \Sigma^\omega$, we call $\rho \in Q^{1+|w|}$ the *run* of $\mathcal{S}$ on w starting in q if $\rho(0) = q$ and for all i, $\rho(i+1) = \delta(\rho(i), w(i))$.

Definition 1.1.18. Let $\mathcal{S} = (Q, \Sigma, \delta)$ be a deterministic transition structure. We define $\delta^* : Q \times \Sigma^* \to Q$ as $\delta^*(q, \varepsilon) = q$ and $\delta^*(q, w \cdot a) = \delta(\delta^*(q, w), a)$.

Definition 1.1.19. Let $\mathcal{S} = (Q, \Sigma, \delta)$ be a deterministic transition structure. For a set $\Omega \subseteq Q^* \cup Q^\omega$, we say that $\mathcal{S}$ has acceptance condition Ω.

We say that a run ρ of $\mathcal{A}$ on some $w \in \Sigma^*$ is *accepting*, if $\rho \in \Omega$; otherwise, the run is *rejecting*. In either case, we say that $\mathcal{A}$ accepts or rejects w.

The *language* of $\mathcal{A}$ with Ω from $q \in Q$ is the set of all words and ω-words that are accepted by $\mathcal{A}$ from q.

Definition 1.1.20. Let $\mathcal{S} = (Q, \Sigma, \delta)$ be a deterministic transition structure. A *strongly connected component*, or SCC, is a set $S \subseteq Q$ such that for all $p, q \in S$, there is a $w \in \Sigma^*$ with $\delta^*(p, w) = q$.

An SCC S is *trivial* if it contains only one state q and $\delta(q, a) \neq q$ for all $a \in \Sigma$.

Definition 1.1.21. A *deterministic finite automaton* (or DFA) is a tuple $\mathcal{A} = (Q, \Sigma, \delta, F)$, where $F \subseteq Q$, such that (Q, Σ, δ) is a deterministic transition structure and has acceptance condition $\Omega = \{\rho \in Q^* \mid \rho(|\rho| + 1) \in F\}$. For the language of (Q, Σ, δ) with Ω from q, we write $L(\mathcal{A}, q)$.

Definition 1.1.22. A *deterministic parity automaton* (or DPA) is a tuple $\mathcal{A} = (Q, \Sigma, \delta, c)$, where $c : Q \to \mathbb{N}$, such that (Q, Σ, δ) is a deterministic transition structure and has acceptance condition $\Omega = \{\rho \in Q^* \mid \min \mathrm{Inf}(c(\rho)) \text{ is even}\}$. For the language of (Q, Σ, δ) with Ω from q, we write $L(\mathcal{A}, q)$.

We call the DPA a *Büchi automaton* (or DBA) if $c(Q) \subseteq \{0, 1\}$. In that case, we use F instead of c.

Definition 1.1.23. Let $\mathcal{A} = (Q, \Sigma, \delta, c)$ be a DPA. We define $c^* : Q \times (\Sigma^* \cup \Sigma^\omega) \to (\mathbb{N}^* \cup \mathbb{N}^\omega)$ as $c^*(q, w) : 1 + |w| \to \mathbb{N}, i \mapsto c(\delta^*(q, w[0, i]))$.

Definition 1.1.24. Let $\mathcal{S} = (Q, \Sigma, \delta)$ be a deterministic transition structure and $P \subseteq Q$. We define $\mathcal{S} \restriction_P = (P, \Sigma, \delta')$ as a transition structure in which δ' might only be a partial function. We set $\delta'(p, a) = \delta(p, a)$ if $\delta(p, a) \in P$, or undefined otherwise.

1.2 Experimental Setup

Even though the focus of this thesis lies in theoretical research, we consider it important to at least roughly analyze the practical "real world" efficiency of ever algorithm. All of them were implemented and tested on a set of deterministic parity automata.

The programming language of choice was C++14. The source code can be found in [24]. The computer used to run the tests was an Arch Linux 4.19.4 64 bit machine powered by an AMD Ryzen 5 1600 processor and 16 GB DDR4-2400 RAM.

Several automata generated randomly using different parameters were used in the testing process. Three major different techniques of generation were used:

1. Use Spot ([5]) to generate a random DPA. (called gendet)
2. Use Spot to generate a random non-deterministic Büchi automaton and use Spot again to convert it to a DPA. (called detspot)
3. Use Spot to generate a random non-deterministic Büchi automaton and use nbautils ([16]) to convert it to a DPA. (called detnbaut)

Figures 1.1, 1.2, 1.3, 1.4 and 1.5 present some information about the automata. Regarding the number of states we stopped the generation at about 150 states, as most algorithms become too slow at that point.

For the detnbaut set, some proecdures can be sped up which is why the upper limit is higher. This is because the nbautils tool internally computes an equivalence relation that implies language equivalence during the determinization. The relation lets us skip the explicit computation of language equivalent states. We refer to [19] and [17] for more information.

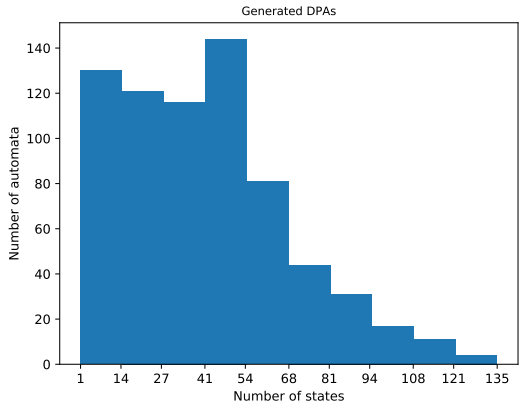
Regarding the plots for stats other than the number of states, we need to mention that we removed far outliers from the data set. For example, the x axis of the SCC plots only ranges up to 10 even though there was as a single example with close to 100 SCCs. Showing these data points would only obscure the information of the plot though.

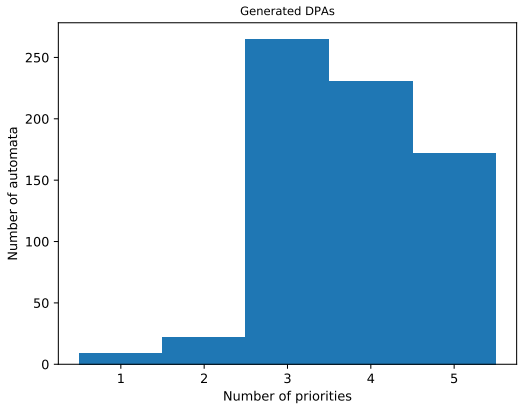
The number of priorities is rather small in general. For gendet, we intentionally limited the number to 5 to mimic real world behavior that can be found in the detnbaut set. In comparison to nbautils, Spot does not perform priority reduction on the determinization result which explains why the detspot automata use more priorities in general.

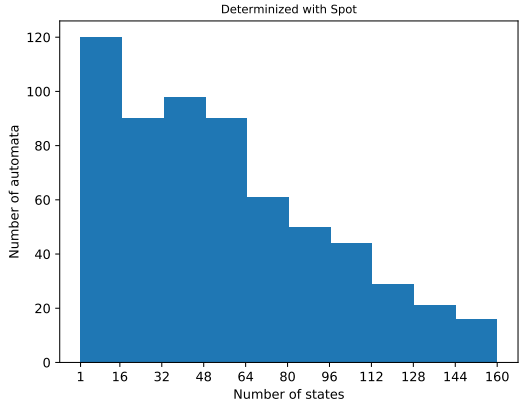
The number of SCCs is consistently small among all three sets. As said before, detspot and gendet contain a few examples of automata with up to 100 SCCs but these are extreme outliers. This low number of SCCs is important to consider, as multiple of the algorithms such as the skip merger perform better the less connected the automaton is.

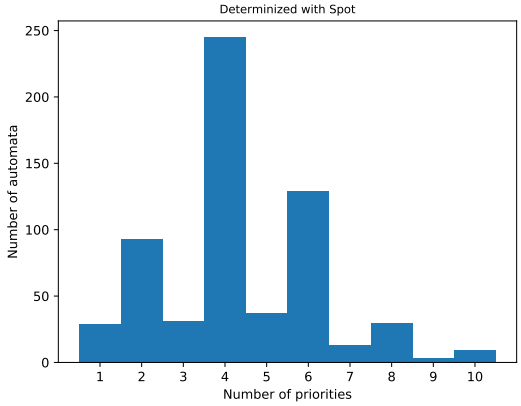
Finally, the average size of equivalence classes $\mathfrak{C}(\equiv_L)$ and the number of classes with more than one element. Again, this is of relevance to some of the reduction algorithms such as LSF as only states which are language equivalent can be merged. We can observe that the gendet set almost entirely consists of trivial classes (i.e. classes of size 1) while detspot and detnbaut show more promise in that regard.

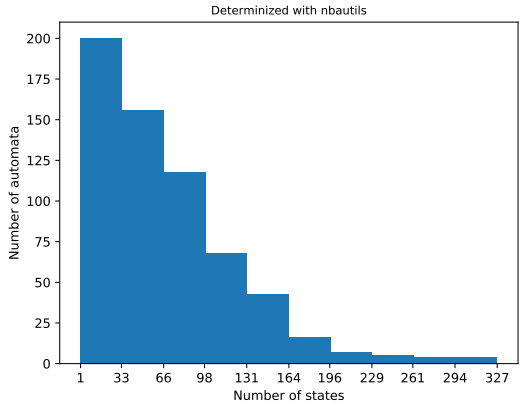
Besides the three classes of data sets that we described above, we also looked at special families of automata that are specifically designed to be difficult to minimize; see [13] and [23]. Unfortunately, these automata proved to be too difficult for our procedures and almost no algorithm could achieve any reduction on an automaton of any size. Thus, we will not discuss these results in any more detail in the upcoming chapters.











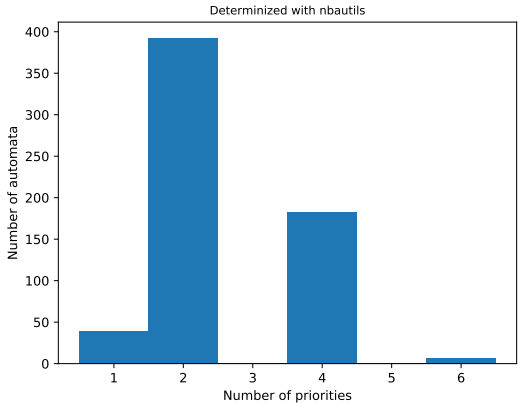
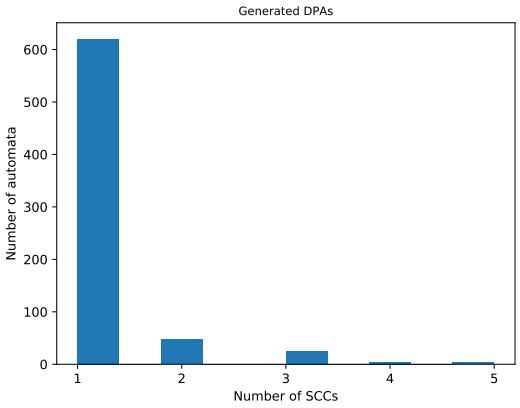
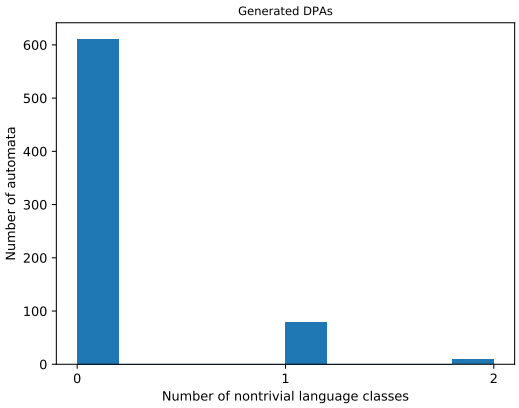
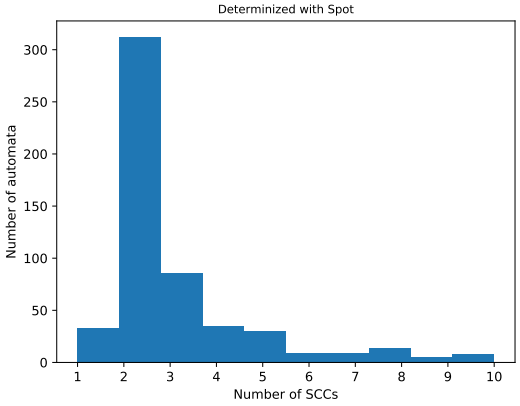


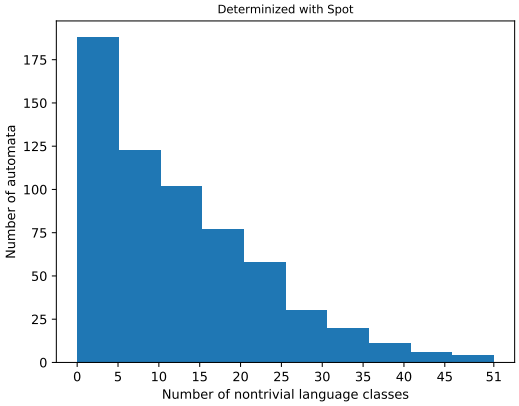
Figure 1.1: Sizes of the automata in the testing environment.

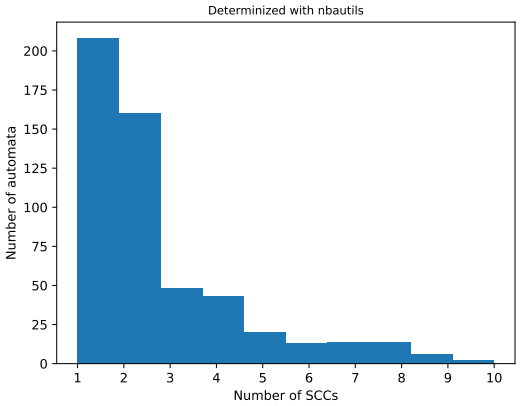
Figure 1.2: Number of priorities in the automata in the testing environment.











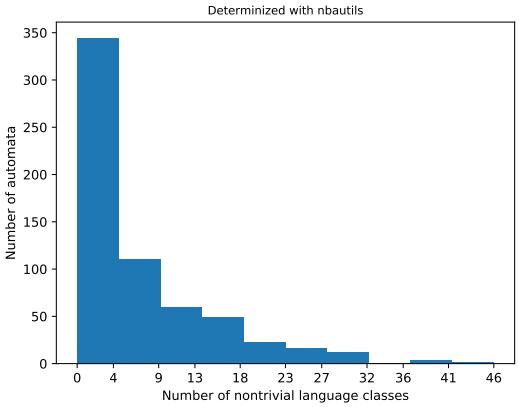


Figure 1.3: Number of SCCS in the automata in the testing environment.

Figure 1.4: Number of classes in $\mathfrak{C}(\equiv_L)$ that contain more than one state.

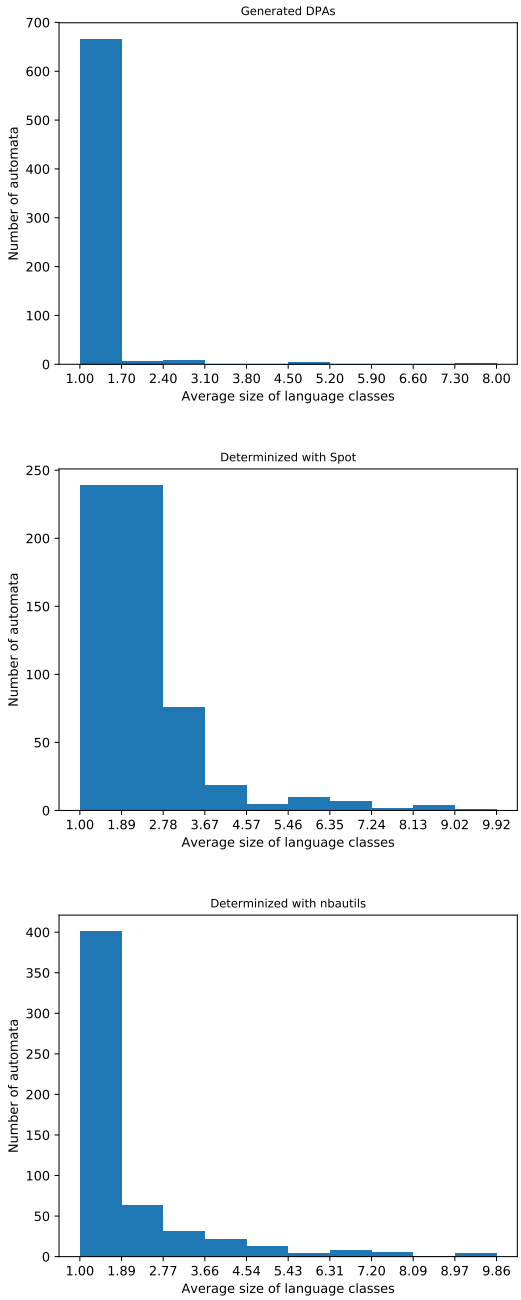


Figure 1.5: Average size of $\equiv_L$-classes of the automata in the testing environment.

Chapter 2

General Theory

2.1 Equivalence Relations

In general, we use symbols $\equiv$, $\sim$, and $\approx$ to denote equivalence relations, mostly between states of an automata. We have already defined the meaning of a general equivalence relation. The type most frequent in this thesis is that which compares pairs of automata and states.

For example, we generally consider two automata or transition structures $\mathcal{A} = (Q_1, \Sigma, \delta_1)$ and $\mathcal{B} = (Q_2, \Sigma, \delta_2)$ with states $p \in Q_1$ and $q \in Q_2$ and consider the question $(\mathcal{A}, p) \equiv (\mathcal{B}, q)$.

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 2.1.1. Given an equivalence relation on states, we use the notation

$$\mathfrak{C}(\sim, \mathcal{A}) = \{\{p \in Q \mid p \sim q\} \mid q \in Q\}$$

for the set of equivalence classes in $\mathcal{A}$.

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 chapter 4.
- Delayed simulation equivalence with resets, $\equiv_{\mathrm{deR}}$. Defined in chapter 4.
- Iterated Moore equivalence, $\equiv_{\mathrm{IM}}$. Defined in chapter 5.
- Path refinement equivalence, $\equiv_{\mathrm{PR}}$. Defined in chapter 6.

- Threshold Moore equivalence, $\equiv_{\mathrm{TM}}^{\sim}$. Defined in chapter 7.
- Labeled SCC filter equivalence, $\equiv_{\mathrm{LSF}}^{k,\sim}$. Defined in chapter 8.

2.1.1 Congruence Relations

Definition 2.1.2. Let $\sim$ be an equivalence relation. We call $\sim$ a *congruence relation* if for all $(\mathcal{A}, p) \sim (\mathcal{B}, q)$ and all $a \in \Sigma$, also $(\mathcal{A}, \delta_1(p, a)) \sim (\mathcal{B}, \delta_2(q, a))$.

Definition 2.1.3. Let $\sim$ be an equivalence relation. We define the *congruence refinement* of $\sim$ as $\approx$ with $(\mathcal{A}, p) \approx (\mathcal{B}, q)$ iff for all $w \in \Sigma^*$, $(\mathcal{A}, \delta_1^*(p, w)) \sim (\mathcal{B}, \delta_2^*(q, w))$.

Lemma 2.1.1. *For a given equivalence relation $\sim$, the congruence refinement of $\sim$ is a congruence relation.*

Proof. Assume towards a contradiction that there are $(\mathcal{A}, p) \approx (\mathcal{B}, q)$ such that $(\mathcal{A}, \delta_1(p, a)) \not\approx (\mathcal{B}, \delta_2(q, a))$. Thus, there is a $w \in \Sigma^*$ with $(\mathcal{A}, \delta_1^*(\delta_1(p, a), w)) \not\sim (\mathcal{B}, \delta_2(\delta_2(q, a), w))$. This means $(\mathcal{A}, \delta_1^*(p, aw)) \not\sim (\mathcal{B}, \delta_2(q, aw))$ and therefore $(\mathcal{A}, p) \not\approx (\mathcal{B}, q)$. □

Lemma 2.1.2. *Let $\sim$ be an equivalence relation and let $\approx$ be its congruence refinement. Then $\approx \subseteq \sim$.*

Proof. If $(\mathcal{A}, p) \approx (\mathcal{B}, q)$, then $(\mathcal{A}, \delta_1^*(p, w)) \sim (\mathcal{B}, \delta_2^*(q, w))$ for all w, in particular $w = \varepsilon$. □

Lemma 2.1.3. *For a given equivalence relation $\sim$, the congruence refinement on a single automaton $\mathcal{A}$ can be computed in $\mathcal{O}(|\mathcal{A}| \cdot \log |\mathcal{A}|)$.*

Proof. We refer to [9], which describes a special case where $p \sim q$ iff $c(p) = c(q)$. □

2.1.2 Language Equivalence

Definition 2.1.4. 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 $L(\mathcal{A}, p) = L(\mathcal{B}, q)$.

Lemma 2.1.4. *$\equiv_L$ 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 2.1.5. *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 [8].

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 $\mathcal{O}(|Q|^2 \cdot |c(Q)|^2)$. □

2.1.3 Priority Almost Equivalence

Definition 2.1.5. Let $\mathcal{A} = (Q_1, \Sigma, \delta_1, c_1)$ and $\mathcal{B} = (Q_2, \Sigma, \delta_2, c_2)$ be DPAs. We define *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 2.1.6. *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 2.1.6. Let $\mathcal{A} = (Q_1, \Sigma, \delta_1, c_1)$ and $\mathcal{B} = (Q_2, \Sigma, \delta_2, c_2)$ be DPAs. We define 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 2.1.7. *$\mathcal{A} \intercal \mathcal{B}$ 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 2.1.8. *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 2.1.9. *Priority almost equivalence of a given DPA can be computed in quadratic time.*

Proof. By Lemma 2.1.7, 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. □

2.1.4 Moore Equivalence

Definition 2.1.7. 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 $w \in \Sigma^*$, $c_1(\delta_1^*(p, w)) = c_2(\delta_2^*(q, w))$.

Lemma 2.1.10. *Moore equivalence is the congruence refinement of $\sim$ with $(\mathcal{A}, p) \sim (\mathcal{B}, q)$ iff $c(p) = c(q)$.*

Corollary 2.1.11. *$\equiv_M$ is a congruence relation.*

Corollary 2.1.12. *Moore equivalence of a given DPA can be computed in log-linear time.*

Theorem 2.1.13. $\equiv_M \subseteq \equiv_\dagger \subseteq \equiv_L$

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, $(\mathcal{A}, q_1) \equiv_L (\mathcal{B}, q_2)$. □

2.2 Representative Merge

Definition 2.2.1. Let $\mathcal{A} = (Q, \Sigma, \delta, c)$ be a DPA and let $\emptyset \neq C \subseteq M \subseteq Q$. Let $\mathcal{A}' = (Q', \Sigma, \delta', c')$ be another DPA. We call $\mathcal{A}'$ a *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, $\delta'(p, a) = q$.

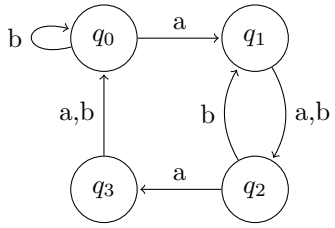


Figure 2.1: Example for representative merges.

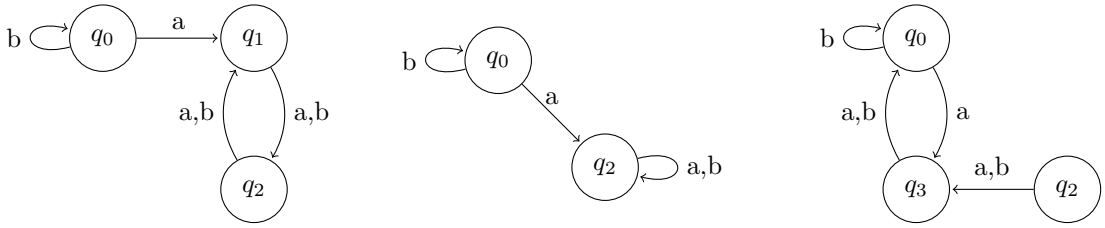


Figure 2.2: Example for representative merges. (after the merge)

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

Definition 2.2.2. Let $\mathcal{A} = (Q, \Sigma, \delta, c)$ be a DPA and let $\mu : D \to (2^Q \setminus \emptyset)$ be a function for some $D \subseteq 2^Q$. We call μ a *merger function* if

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

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

Figure 2.1 shows an example automaton. We know want to build a representative merge of this automaton w.r.t. $M = \{q_1, q_3\}$ by candidates $C = \{q_1, q_2, q_3\}$. The three possible automata (one for each choice of $r_M \in C$) are shown in figure 2.2.

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

Lemma 2.2.1. *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. □

Lemma 2.2.2. *Given a DPA $\mathcal{A}$ and a merger function $\mu : D \to 2^Q$ in suitable data structures, a representative merge of $\mathcal{A}$ w.r.t. μ can be computed in linear time.*

Proof. If for each $q \in Q$, we can find the $M \in D$ with $q \in M$ in constant time, and are also able to evaluate $\mu(M)$ in constant time (e.g. with a hash map), then building the representative merge comes down to iterating over all states in the automaton and checking their status in the sets in D or their candidates. This can be done in linear time. □

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

Lemma 2.2.3. *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, $\mu_{\div}^{\sim}$. Defined below.
- Moore merger, μ_M. Defined below.
- Skip merger, $\mu_{\text{skip}}^{\sim}$. Defined in chapter 3.
- Delayed simulation merger, μ_{de}. Defined in chapter 4.
- Iterated Moore merger, μ_{IM}. Defined in chapter 5.
- Path refinement merger, $\mu_{\text{PR}}^{\lambda}$. Defined in chapter 6.
- Treshold Moore merger, $\mu_{\text{TM}}^{\sim}$. Defined in chapter 7.
- Labeled SCC Filter merger, $\mu_{\text{LSF}}^{k,\sim}$. Defined in chapter 8.

2.2.1 Quotient merger

Definition 2.2.3. Let $\sim$ be a congruence relation. We define the *quotient merger*

$$\mu_{\div}^{\sim} : \mathfrak{C}(\sim, \mathcal{A}) \to 2^Q, \kappa \mapsto \kappa$$

.

Lemma 2.2.4. *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. $\mu_{\div}^{\sim}$.*

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. □

2.2.2 Moore merger

Definition 2.2.4. We define the special *Moore merger* $\mu_M = \mu_{\dot{=}}^{\equiv_M}$.

Lemma 2.2.5. *Let $\mathcal{A}$ be a DPA and let $\mathcal{A}'$ be a representative merge of $\mathcal{A}$ w.r.t. μ_M. Then $\mathcal{A} \equiv_L \mathcal{A}'$.*

Proof. Directly implied by Lemma 2.2.4 and Theorem 2.1.13. □

The representative merge of a DPA w.r.t. μ_M is unique up to isomorphism and will actually be the same automaton that is created from Hopcroft's reduction algorithm. ([9])

In figure 2.3, the efficiency of μ_M on different types of automata is shown. On the gendet and detnbaut classes, barely any state reduction is achieved. gendet lacks patterns in its automata and detnbaut already has certain minimization techniques applied during generation. detspot on the other hand shows some small but noticeable reduction. In figure 2.5, we can see that the number of removed states is roughly proportional to the number of total states, which is no big surprise.

As is to be expected from Corollary 2.1.12 and confirmed in figure 2.4, the Moore merger is very easy to compute.

2.3 Reachability

Definition 2.3.1. Let $\mathcal{S} = (Q, \Sigma, \delta)$ be a deterministic transition structure. We define the *reachability order* $\preceq_{\text{reach}}^{\mathcal{S}}$ as $p \preceq_{\text{reach}}^{\mathcal{S}} q$ if and only if q is reachable from p.

We want to note here that we always assume for all automata to be completely connected, 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.

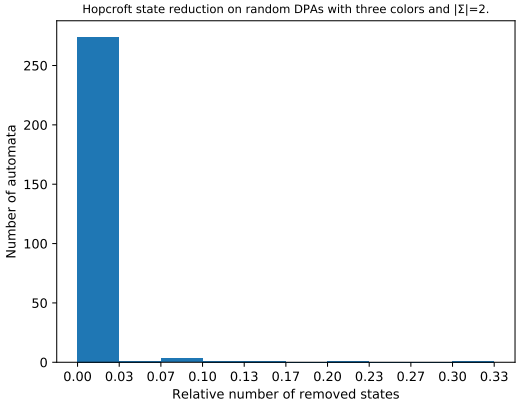
Lemma 2.3.1. *$\preceq_{reach}^{\mathcal{S}}$ is a preorder.*

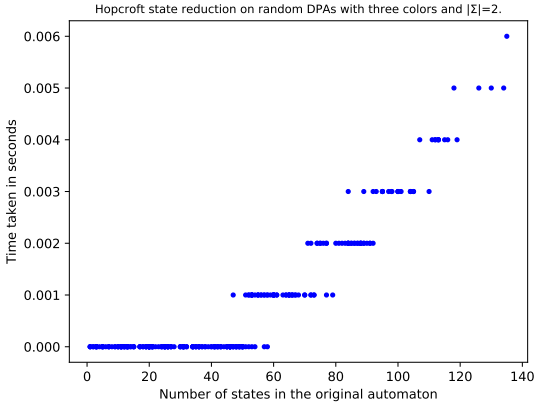
Definition 2.3.2. Let $\mathcal{S} = (Q, \Sigma, \delta)$ be a deterministic transition structure. We call a relation $\preceq$ a *total extension of reachability* if it is a minimal superset of $\preceq_{\text{reach}}^{\mathcal{S}}$ that is also a total preorder.

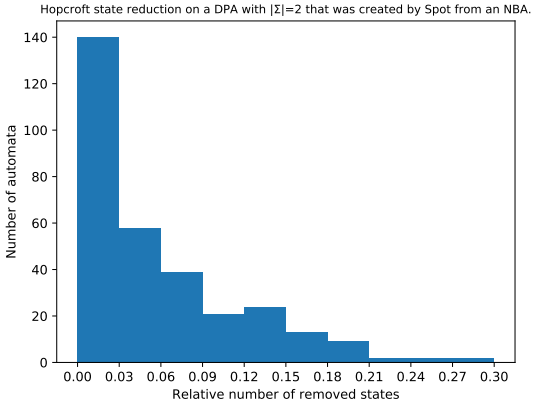
For $p \preceq q$ and $q \preceq p$, we write $p \simeq q$.

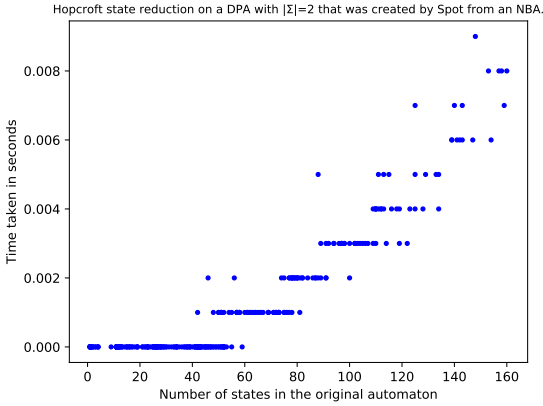
Lemma 2.3.2. *For a given deterministic transition structure $\mathcal{S}$, a total extension of reachability is computable in $\mathcal{O}(|\mathcal{S}|)$.*

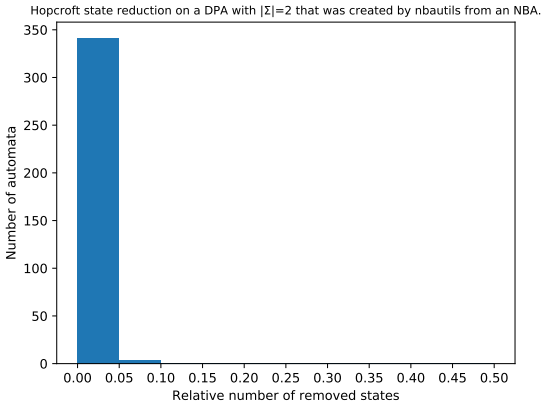
Proof. Using e.g. Kosaraju's algorithm [21], the SCCs of $\mathcal{A}$ can be computed in linear time. We can now build a DAG from $\mathcal{A}$ by merging all states in an SCC into a single state; iterate over all transitions (p, a, q) and add an a-transition from the merged representative of p to that of q.











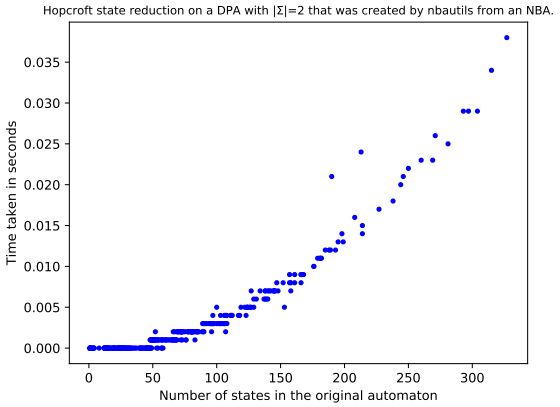


Figure 2.3: State reduction of different automata using μ_M.

Figure 2.4: Time of the state reduction of different automata using μ_M.

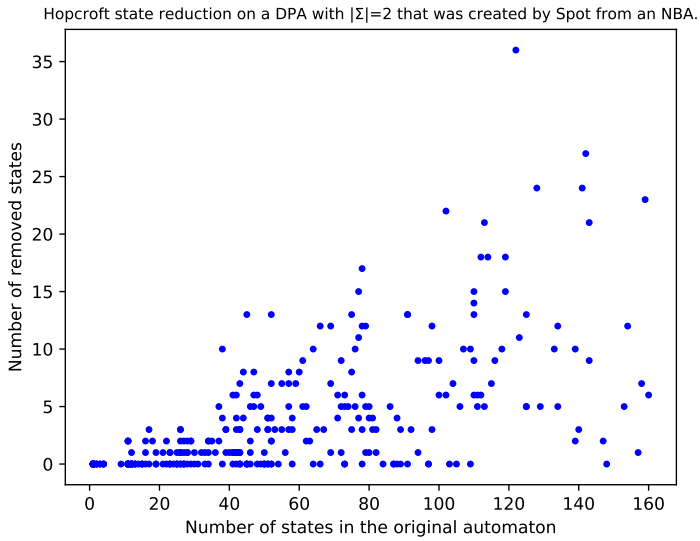


Figure 2.5: State reduction of detspot automata using μ_M.

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. □

2.4 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 2.4.1. *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, $\operatorname{Inf} c(\rho) = \operatorname{Inf} c'(\rho')$ 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 2.4.1. 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.

Not only does this algorithm cause the automaton to be c-normalized, it also makes sure that the number of priorities used is minimal on the given transition structure. Again, we refer to the original publication for more information.

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
```

Chapter 3

Skip Merger

3.1 Theory

The so called *skip merger* we introduce now uses the rather simple idea that if we consider classes of language equivalent states, then we only need to use the "last" SCC of those states and can remove all others.

Definition 3.1.1. Let $\mathcal{A} = (Q, \Sigma, \delta, c)$ be a DPA and let $\sim \subseteq Q \times Q$ be a congruence relation on $\mathcal{A}$. Let $\preceq \subseteq Q \times Q$ be a total extension of $\preceq^{\mathcal{A}}_{\text{reach}}$.

We define the *skip merger function* $\mu^{\sim}_{\text{skip}} : D \to 2^Q$ as follows: for each equivalence class $\kappa \in \mathfrak{C}(\sim, \mathcal{A})$, let $C_\kappa \subseteq \kappa$ be the set of $\preceq$-maximal elements in κ. Let $M_\kappa = \kappa \setminus C_\kappa$. Then we have $D = \{M_\kappa \mid \kappa \in \mathfrak{C}(\sim, \mathcal{A})\}$ and $\mu^{\sim}_{\text{skip}}(M_\kappa) = C_\kappa$.

Now that we have established the definition of the merger, we want to analyze its structure and prove its correctness. For the rest of this section, we use $\mathcal{A} = (Q, \Sigma, \delta, c)$ as a DPA, $\sim$ as a congruence relation, and $\mathcal{B} = (Q_{\mathcal{B}}, \Sigma, \delta_{\mathcal{B}}, c_{\mathcal{B}})$ as a representative merge of $\mathcal{A}$ w.r.t. $\mu^{\sim}_{\text{skip}}$.

Lemma 3.1.1. *Let ρ be a run on α in $\mathcal{B}$. Then for all i, $\rho(i) \preceq \rho(i+1)$. Furthermore, we have $\rho(i) \prec \rho(i+1)$ if and only if $\rho(i) \prec r_{\delta(\rho(i),\alpha(i))}$ (with $r_q = r_{M_{[q]_\sim}}$).*

Proof. Let i be an arbitrary index of the run. If $\rho(i)$ to $\rho(i+1)$ is also a transition in $\mathcal{A}$, then $\rho(i+1)$ is reachable from $\rho(i)$ in $\mathcal{A}$ and hence $\rho(i) \preceq \rho(i+1)$ by definition of the preorder. Otherwise the transition used was redirected in the construction. The way the redirection is defined, this implies $\rho(i) \prec \rho(i+1)$.

We move on to the second part of the lemma. If $\rho(i) \prec r_{M_{[\delta(\rho(i),\alpha(i))]_\sim}}$, then the transition is redirected to $\rho(i+1) = r_{\delta(\rho(i),\alpha(i))}$ and the statement holds.

For the other direction, let $\rho(i) \prec \rho(i+1)$ and assume towards a contradiction that $\rho(i) \not\prec r_{\delta(\rho(i),\alpha(i))}$. This means that the transition was not redirected and $\rho(i+1) = \delta(\rho(i), \alpha(i))$. Since $\preceq$ is total, we have $r_{\delta(\rho(i),\alpha(i))} = r_{\rho(i+1)} \preceq \rho(i) \prec \rho(i+1)$ which contradicts the $\preceq$-maximality of representatives. □

Lemma 3.1.2. *Let $p, q \in Q_{\mathcal{B}}$. If $p \sim q$, then p and q lie in the same SCC.*

Proof. It suffices to restrict ourselves to $q = r_{[q]_\sim} = r_{[p]_\sim}$. If we can prove the Lemma for this case, then the general statement follows by transitivity.

Let p_0 be a state from which both p and q are reachable. Let $p_0 \cdots p_n$ be a minimal run of $\mathcal{B}$ that reaches p. By Lemma 3.1.1, we have $p_0 \preceq \cdots \preceq p_n$. Whenever $p_i \prec p_{i+1}$, a redirected transition to the representative $r_{[p_{i+1}]_\sim} = p_{i+1}$ is taken.

Let k be the first position after which no redirected transition is taken anymore. For the first case, assume that $k < n$. Then $p_i \simeq r_{[p_{i+1}]_\sim}$ for all $i \geq k$. In particular, $p_{n-1} \simeq q$. Since $p_{n-1} \preceq p_n$, we also have $q \preceq p_n$. The representatives are chosen $\preceq$-maximal in their $\sim$-class, so $q \simeq p_n$.

The second case is $k = n$. In that case, the transition from p_{n-1} to p_n is redirected and $p_n = r_{[p_n]_\sim} = q$. □

Lemma 3.1.3. *Let $\rho \in Q^\omega$ be an infinite run in $\mathcal{B}$. Then ρ has a suffix that is a run in $\mathcal{A}$.*

Proof. We show that only finitely often a redirected transition is used in ρ. Then, from some point on, only transitions that also exist in $\mathcal{A}$ are used. The suffix starting at this point is the run that we are looking for.

Let $\rho = p_0 p_1 \cdots$. By Lemma 3.1.1, we have $p_i \preceq p_{i+1}$ for all i and $p_i \prec p_{i+1}$ whenever a redirected transition is taken. As Q is finite, we can only move up in the order finitely often. This proves our claim. □

Theorem 3.1.4. *Let $\sim \subseteq \equiv_L$. Then $\mathcal{A}$ and $\mathcal{B}$ are language equivalent.*

Proof. Let $\alpha \in \Sigma^\omega$ be a word and let ρ be of $\mathcal{B}$ starting in q_0 on α. By Lemma 3.1.3, ρ has a suffix π which is a run segment of $\mathcal{A}$ on some suffix β of α. The acceptance condition of DPAs is prefix independent, so $\alpha \in L(\mathcal{B}, q_0)$ iff ρ is an accepting run iff π is an accepting run. Since the priorities do not change during the construction, π is accepting in $\mathcal{B}$ iff it is accepting in $\mathcal{A}$.

Let $w \in \Sigma^*$ be the prefix of α with $\alpha = w\beta$. By Lemma 2.2.3, we know that $(\mathcal{A}, \delta^*(q_0, w)) \sim (\mathcal{A}, \delta^*_{\mathcal{B}}(q_0, w))$. Since every state is $\sim$-equivalent to its representative and $\sim$ is a congruence relation, we also know $\delta^*_{\mathcal{B}}(q_0, w) \sim \delta^*_{\mathcal{B}}(r_{[q_0]_\sim}, w)$. From $\delta^*_{\mathcal{B}}(r_{[q_0]_\sim}, w)$, the run π accepts β iff $\alpha \in L(\mathcal{B}, q_0)$. As $\sim$ implies language equivalence, the same must hold for $\delta^*_{\mathcal{A}}(q_0, w)$. Therefore, $\alpha \in L(\mathcal{A}, q_0)$ iff $\alpha \in L(\mathcal{B}, q_0)$. □

3.2 Computation

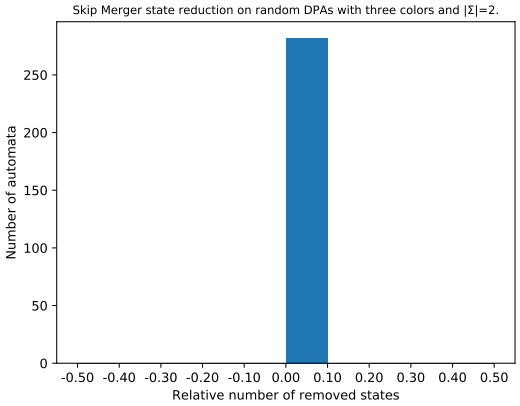
Lemma 3.2.1. *For a given $\mathcal{A}$ and $\sim$, $\mu^{\sim}_{skip}$ can be computed in $\mathcal{O}(|\mathcal{A}|)$.*

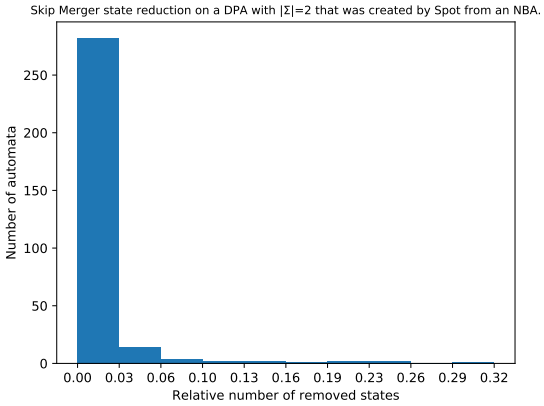
Proof. As seen in Lemma 2.3.2, $\preceq$ can be computed in linear time. Assuming that $\sim$ is given by a suitable data structure, each equivalence class can easily be accessed and $\preceq$-maximal elements can be found in linear time. □

3.3 Efficiency

As we can see in figures 3.1 and 3.2, barely any reduction occurs using the skip merger function, even when using the "best" equivalence relation $\equiv_L$. This can be explained by the restrictive nature of the procedure and the fact that our testing automata are in general well connected. For two states to be merged, they have to lie in different SCCs, meaning that the fewer SCCs an automaton

has, the less likely it is to be affected by the skip merger. As we saw in figure 1.3, most automata have at most three different SCCs.





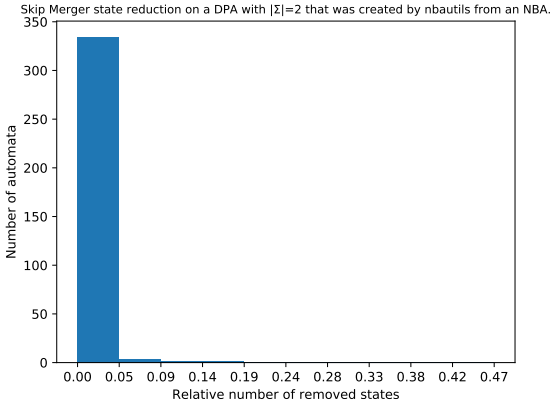
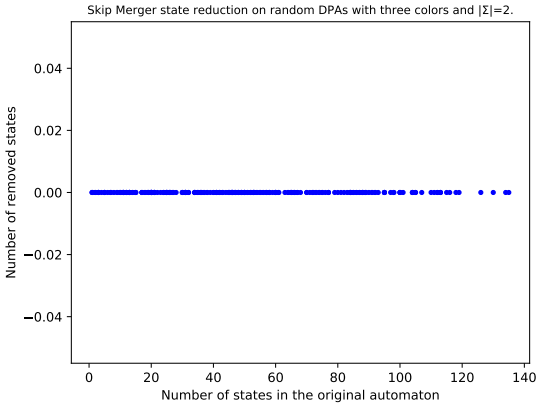
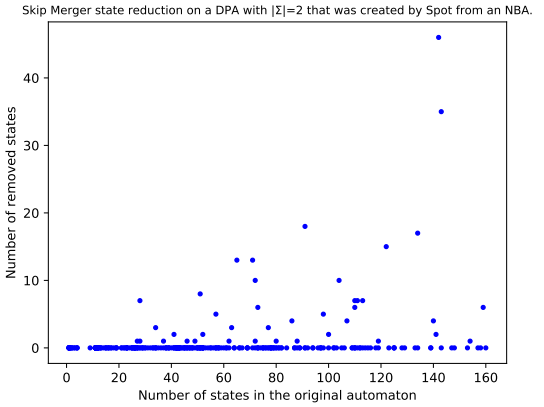


Figure 3.1: State reduction of different automata using $\mu_{\text{skip}}^{\equiv_L}$.





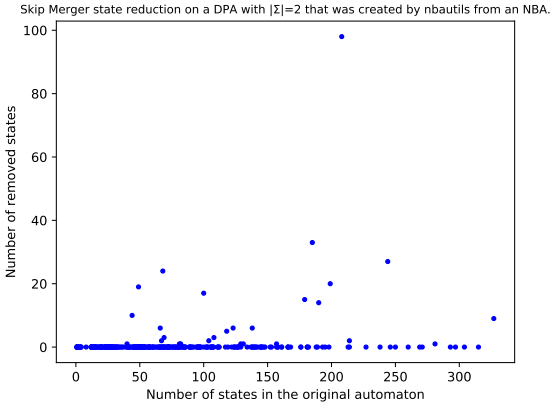


Figure 3.2: State reduction of different automata using $\mu_{\text{skip}}^{\equiv_L}$.

Chapter 4

Delayed Simulation

4.1 Theory

In this part we consider delayed simulation and variants thereof on DPAs. This approach is based on [7], which considered the simulation for alternating parity automata. The DPAs we use are a special case of these APAs and therefore worth examining.

Definition 4.1.1. Let $\mathcal{A} = (Q_1, \Sigma, \delta_1, c_1)$ and $\mathcal{B} = (Q_2, \Sigma, \delta_2, c_2)$ be DPAs. For $\alpha \in \Sigma^\omega$, we write $(\mathcal{A}, p) \leq^{\alpha}_{\mathrm{de}} (\mathcal{B}, q)$ iff the following holds:

Let π and ρ be the runs of $\mathcal{A}$ and $\mathcal{B}$ on α starting in p and q respectively. For every position i, there is a $j \geq i$ such that $\pi(i) \leq \min\{c_1(p'), c_2(q')\}$ is odd or $\rho(i) \leq \min\{c_1(p'), c_2(q')\}$ is even, where $p' = \pi(i)$ and $q' = \rho(i)$.

We define *delayed simulation* as $(\mathcal{A}, p) \leq_{\mathrm{de}} (\mathcal{B}, q)$ iff $(\mathcal{A}, p) \leq^{\alpha}_{\mathrm{de}} (\mathcal{B}, q)$ is true for every α.

This definition might seem arbitrary. It is adapted to the result of delayed simulation games from [7] that we describe later on. A more intuitive descendant of $\leq_{\mathrm{de}}$ is delayed simulation equivalence $\equiv_{\mathrm{de}}$ that is defined below.

Definition 4.1.2. We define $(\mathcal{A}, p) \equiv_{\mathrm{de}} (\mathcal{B}, q)$ iff both $(\mathcal{A}, p) \leq_{\mathrm{de}} (\mathcal{B}, q)$ and $(\mathcal{B}, q) \leq_{\mathrm{de}} (\mathcal{A}, p)$.

Lemma 4.1.1. *$(\mathcal{A}, p) \equiv_{de} (\mathcal{B}, q)$ if and only if the following property holds for all $w \in \Sigma^*$:*

Let $p' = \delta_1^*(p, w)$ and $q' = \delta_2^*(q, w)$. Every run that starts in p' or q' eventually sees a priority less than or equal to $\min\{c_1(p'), c_2(q')\}$.

Proof. **If** We show that $(\mathcal{A}, p) \leq_{\mathrm{de}} (\mathcal{B}, q)$. The other inclusion follows by symmetry. Let $\alpha \in \Sigma^\omega$ and let π and ρ be the runs on α starting in p and q. Let i be a position, $p' = \pi(i)$, and $q' = \rho(i)$. Assume that there is no $j \geq i$ such that $\pi(j)$ is odd and $c_1(\pi(j)) \leq \min\{c_1(p'), c_2(q')\}$. We will show that there must be a $j \geq i$ where $c_2(\rho(j)) \leq \min\{c_1(p'), c_2(q')\}$ is even.

By assumption of the Lemma, there is a j with $\rho(j) \leq \min\{c_1(p'), c_2(q')\}$. If any of these j is even, we are done; otherwise, they are all odd. Let j be chosen such that $\rho(j)$ is minimal. There must be a $k \geq j$ such that $c_1(\pi(k)) \leq \min\{c_1(\pi(j)), c_2(\rho(j))\}$. This value cannot be odd, as $c_1(\pi(k)) \leq c_2(\rho(j)) \leq \min\{c_1(p'), c_2(q')\}$ happens only at even values. Hence, it must also be strictly smaller than $c_2(\rho(j))$.

We can apply the assumption one last time, to find a position $l \geq k$ such that $c_2(\rho(l)) \leq \min\{c_1(\pi(k)), c_2(\rho(k))\} < c_2(\rho(j))$ which contradicts the choice of j to have minimal priority.

Only If Assume $(\mathcal{A}, p) \equiv_{\text{de}} (\mathcal{B}, q)$. We show that the described property is true. Towards a contradiction, assume that there are w and β such that from p' and q' one of the runs π and ρ only visits priorities greater than $\min\{c_1(p'), c_2(q')\}$. Without loss of generality, let that run be π.

Let $i > |w|$ be a position such that $c_2(\rho(i))$ becomes minimal. If that value is odd, consider $(\mathcal{A}, p) \leq_{\text{de}} (\mathcal{B}, q)$; there must be a position $j \geq i$, such that $c_1(\pi(j)) \leq \min\{c_1(\pi(i)), c_2(\rho(i))\}$ is odd or $c_2(\rho(j)) \leq \min\{c_1(\pi(i)), c_2(\rho(i))\}$ is even. The first case cannot happen, as then π would visit a priority less than $\min\{c_1(p'), c_2(q')\}$. The second case however cannot be true either, as $c_2(\rho(i))$ is odd and there is no priority in $c_2(\rho)$ smaller than that value.

If $c_2(\rho(i))$ would be even, we could use $(\mathcal{B}, q) \leq_{\text{de}} (\mathcal{A}, p)$ to find a similar contradiction. □

Lemma 4.1.2. *$\equiv_{de}$ is a congruence relation.*

Proof. Using the characterization of $\equiv_{\text{de}}$ in Lemma 4.1.1 easily shows that it is an equivalence relation.

Now to show that $\equiv_{\text{de}}$ is also a congruence relation: Let $(\mathcal{A}, p) \equiv_{\text{de}} (\mathcal{B}, q)$ and $a \in \Sigma$. We want to prove that also $(\mathcal{A}, p') \equiv_{\text{de}} (\mathcal{B}, q')$ with $p' = \delta_1(p, a)$ and $q' = \delta_2(q, a)$. Assume that this is not the case, so there is a $w \in \Sigma^*$ such that the following is true for $p'' = \delta_1^*(p', w)$ and $q'' = \delta_2^*(q', w)$: $c_1(p'') < c_2(q'')$ and there is a run starting in q'' that only sees priorities greater than $c_1(p'')$. Then, however, we could also reach p'' and q'' from p and q by the word aw which implies that $(\mathcal{A}, p) \not\equiv_{\text{de}} (\mathcal{B}, q)$. □

We move on to establishing $\equiv_{\text{de}}$ as a "good" relation for our needs and define a fitting merger function.

Lemma 4.1.3. *Let $\mathcal{A}$ be a DPA and let π and ρ be runs of $\mathcal{A}$ on the same word α but starting at different states. If $\pi(0) \equiv_{de} \rho(0)$, then* $\min Occ(c(\pi)) = \min Occ(c(\rho))$.

Proof. Let $\pi(0) = p$ and $\rho(0) = q$. Assume towards a contradiction that the statement is false and $\min \text{Occ}(c(\pi)) < \min \text{Occ}(c(\rho))$. Let $k = \min \text{Occ}(c(\pi))$ and let n be the first position at which $c(\pi(n)) = k$. Let $p' = \pi(n)$ and $q' = \rho(n)$. These two states are reachable from p and q respectively with the word $\alpha[0, n]$.

By Lemma 4.1.1, the run $\rho[n, \omega]$ must eventually see a priority at most k which contradicts the assumption. □

Theorem 4.1.4. *Let $\mathcal{A} = (Q, \Sigma, \delta, c)$ be a DPA and let $p, q \in Q$ with $p \equiv_{de} q$ and $c(p) < c(q)$. Define $\mathcal{A}' = (Q, \Sigma, \delta, c')$ with $c'(s) = \begin{cases} c(p) & \text{if } s = q \\ c(s) & \text{else} \end{cases}$. Then $\mathcal{A} \equiv_L \mathcal{A}'$.*

Proof. Let $q_0 \in Q$ be an arbitrary state. We show that $L(\mathcal{A}, q_0) = L(\mathcal{A}', q_0)$.

First, consider the case that $c(p)$ is an even number. The parity of each state is at least as good in $\mathcal{A}'$ as it is in $\mathcal{A}$, so $L(\mathcal{A}, q_0) \subseteq L(\mathcal{A}', q_0)$. For the other direction, assume there is an $\alpha \in L(\mathcal{A}', q_0) \setminus L(\mathcal{A}, q_0)$, so the respective run $\rho \in Q^\omega$ is accepting in $\mathcal{A}'$ but not in $\mathcal{A}$.

For this to be true, ρ must visit q infinitely often and $c'(q)$ must be the lowest priority that occurs infinitely often; otherwise, the run would have the same acceptance in both automata. Thus, there is a finite word $w \in \Sigma^*$ such that from q, $\mathcal{A}$ reaches again q via w and inbetween only priorities greater than $c'(q)$ are seen.

Now consider the word w^ω and the run π_q of $\mathcal{A}$ on said word starting in q. With the argument above, we know that the minimal priority occuring in $c(\pi_q)$ is greater than $c'(q)$. If we take the run π_p on w^ω starting at p though, we find that this run sees priority $c(p) = c'(q)$ at the very beginning. This contradicts Lemma 4.1.3, as $p \equiv_{\mathrm{de}} q$. Thus, the described α cannot exist.

If $c(p)$ is an odd number, a very similar argumentation can be applied with the roles of $\mathcal{A}$ and $\mathcal{A}'$ reversed. We omit this repetition. □

Together with Lemma 2.2.4, this allows for a merger function that preservers language.

Definition 4.1.3. We define the *delayed simulation merger* as

$$\mu_{\mathrm{de}} : \mathfrak{C}(\equiv_{\mathrm{de}}, \mathcal{A}) \to 2^Q, \kappa \mapsto \{q \in \kappa \mid c(q) = \min c(\kappa)\}.$$

Corollary 4.1.5. *Let $\mathcal{A}$ be a DPA. Then $\mathcal{A}$ is language equivalent to every representative merge w.r.t. μ_{de}.*

Proof. With Theorem 4.1.4, we can construct a language equivalent $\mathcal{A}' = (Q, \Sigma, \delta, c')$ such that for all $\kappa \in \mathfrak{C}(\equiv_{\mathrm{de}})$, all states in κ have the same priority in c'. Every representative merge of $\mathcal{A}$ w.r.t. μ_{de} is a representative merge of $\mathcal{A}'$ w.r.t. μ_{de}.

In $\mathcal{A}'$, μ_{de} is the same as $\mu_{\dot{=}}^{\equiv_{\mathrm{de}}}$. The fact that representative merges of $\mathcal{A}'$ w.r.t. μ_{de} preserve language follows from Lemma 2.2.4. □

4.2 Computation

So far we have given a definition of $\equiv_{\mathrm{de}}$ that provides no obvious way of how to compute that relation. For that we can use delayed simulation games as described in [7].

Definition 4.2.1. We introduce ✓ as an "infinity" to the natural numbers and define the **obligation order** $\leq_✓ \subseteq (\mathbb{N} \cup \{✓\}) \times (\mathbb{N} \cup \{✓\})$ as $0 \leq_✓ 1 \leq_✓ 2 \leq_✓ \cdots \leq_✓ ✓$.

Second, we define an order of "goodness" on parity priorities $\preceq_{\mathrm{p}} \subseteq \mathbb{N} \times \mathbb{N}$ as $0 \preceq_{\mathrm{p}} 2 \preceq_{\mathrm{p}} 4 \preceq_{\mathrm{p}} \cdots \preceq_{\mathrm{p}} 5 \preceq_{\mathrm{p}} 3 \preceq_{\mathrm{p}} 1$.

Definition 4.2.2. 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 *delayed simulation automaton* $(\mathcal{A}, \mathcal{B})_{\mathrm{de}} = (Q_{\mathrm{de}}, \Sigma, \delta_{\mathrm{de}}, F_{\mathrm{de}})$, which is a deterministic Büchi automaton, as follows.

- $Q_{\mathrm{de}} = Q_1 \times Q_2 \times (c_1(Q_1) \cup c_2(Q_2) \cup \{✓\})$, i.e. the states are given as triples in which the first two components are states from $\mathcal{A}$ and the third component is either a priority from $\mathcal{A}$ or $\mathcal{B}$ or ✓.
- The alphabet remains Σ.
- $\delta_{\mathrm{de}}((p, q, k), a) = (p', q', \gamma(c_1(p'), c_2(q'), k)$, where $p' = \delta_1(p, a)$, $q' = \delta_2(q, a)$. The first two components behave like a regular product automaton. γ is defined below.
- $F_{\mathrm{de}} = Q_1 \times Q_2 \times \{✓\}$.

$\gamma : \mathbb{N} \times \mathbb{N} \times (\mathbb{N} \cup \{\checkmark\}) \to \mathbb{N} \cup \{\checkmark\}$ is the update function of the third component and defines the "obligations" as they are called in [7]. It is defined as

$$\gamma(i,j,k) = \begin{cases} \checkmark & \text{if } i \text{ is odd and } i \leq_\checkmark k \text{ and } j \preceq_\mathrm{p} i \\ \checkmark & \text{if } j \text{ is even and } j \leq_\checkmark k \text{ and } j \preceq_\mathrm{p} i \\ \min_{\leq_\checkmark}\{i,j,k\} & \text{else} \end{cases}$$

If $\mathcal{A} = \mathcal{B}$, we simply write $\mathcal{A}_\mathrm{de}$ for the delayed simulation automaton.

Furthermore, for states p in $\mathcal{A}$ and q in $\mathcal{B}$, we define an initial state

$$q_0^\mathrm{de}(p,q) = (p, q, \gamma(c_1(p), c_2(q), \checkmark))$$

Lemma 4.2.1. *Let $\mathcal{A}$ be a DPA and $\rho \in Q_{de}^\omega$ be a run of $\mathcal{A}_\mathrm{de}$ on some word. Let $k \in (\mathbb{N} \cup \{\checkmark\})^\omega$ be the third component during ρ. For all i, $k(i+1) \leq_\checkmark k(i)$ or $k(i+1) = \checkmark$.*

Proof. Follows directly from the definition of γ. □

Lemma 4.2.2. *$(\mathcal{A}, p) \leq_{de}^\alpha (\mathcal{B}, q)$ iff $\alpha \in L((\mathcal{A},\mathcal{B})_{de}, q_0^{de}(p,q))$*

Proof. Let $\pi \in Q_1^\omega$ and $\rho \in Q_2^\omega$ be the runs of $\mathcal{A}$ and $\mathcal{B}$ on α starting in p and q respectively. Let $\sigma \in Q_\mathrm{de}^\omega$ be the run of $(\mathcal{A},\mathcal{B})_\mathrm{de}$ on α starting in $q_0^\mathrm{de}(p,q)$. By definition of the delayed simulation automaton there are γ_i s.t. $\sigma(i) = (\pi(i), \rho(i), \gamma_i)$.

If Assume that σ is an accepting run, i.e. there are infinitely many i such that $\gamma_i = \checkmark$. We now want to show that $(\mathcal{A}, p) \leq_\mathrm{de}^\alpha (\mathcal{B}, q)$ holds. Assume the opposite towards contradiction, so there is an i such that for all $j \geq i$, $\pi(j) > \min\{c_1(\pi(i)), c_2(\rho(i))\}$ or $\pi(j)$ is even, and $\rho(j) > \min\{c_1(\pi(i)), c_2(\rho(i))\}$ or $\rho(j)$ is odd.

We consider the first position $j \geq i$ with $\gamma_j = \checkmark$. By Lemma 4.2.1, $\gamma_i \geq_\checkmark \gamma_{i+1} \geq_\checkmark \cdots \geq_\checkmark \gamma_{j-1}$, and $\gamma_j = \gamma(c_1(\pi(j)), c_2(\rho(j)), \gamma_{j-1})$. Since this evaluates to $\checkmark$, we know that $\pi(j)$ is odd and $\pi(j) \leq_\checkmark \gamma_{j-1} \leq_\checkmark \gamma_i$, or $\rho(j)$ is even and $\rho(j) \leq_\checkmark \gamma_{j-1} \leq_\checkmark \gamma_i$.

If $j \neq i$, then $\gamma_i \leq \min_{\leq_\checkmark}\{c_1(\pi(i)), c_2(\rho(i))\}$ by definition of the γ function. This is a contradiction to our previous assumption that such a position j is never reached.

If $j = i$, then $c_2(\rho(i)) \preceq_p c_1(\pi(i))$, which gives us the same contradiction when unfolding the definition of $\preceq_p$.

Only If Assume that $(\mathcal{A}, p) \leq_\mathrm{de}^\alpha (\mathcal{B}, q)$ is true. Towards a contradiction, assume that σ is not accepting. By Lemma 4.2.1 and because $\leq_\checkmark$ is a well-ordering, there is a position i after which γ_i does not change anymore, meaning $\gamma_{i-1} \neq \gamma_i$ but $\gamma_j = \gamma_{j+1}$ for all $j \geq i$.

As $(\mathcal{A}, p) \leq_\mathrm{de}^\alpha (\mathcal{B}, q)$ is true, let $j \geq i$ be a position such that $c_1(\pi(j)) \leq \min\{c_1(\pi(i)), c_2(\rho(i))\}$ is odd or $c_2(\rho(j)) \leq \min\{c_1(\pi(i)), c_2(\rho(i))\}$ is even. As i was the last position at which γ_i changed, we have $\gamma_i = \min\{c_1(\pi(i)), c_2(\rho(i))\}$. As this value does not change anymore, we also have $\gamma_{j-1} = \min\{c_1(\pi(i)), c_2(\rho(i))\}$. Together with our assumption about position j, this would mean $\gamma_j = \checkmark$, which is a contradiction. □

Corollary 4.2.3. *$(\mathcal{A}, p) \equiv_{de} (\mathcal{B}, q)$ iff $L((\mathcal{A},\mathcal{B})_{de}, q_0^{de}(p,q)) = L((\mathcal{B},\mathcal{A})_{de}, q_0^{de}(q,p)) = \Sigma^\omega$*

Theorem 4.2.4. *For a given DPA $\mathcal{A}$, $\equiv_{de}$ can be computed in $\mathcal{O}(|Q|^2 \cdot |c(Q)|)$.*

Proof. $\mathcal{A}_{\text{de}}$ is of size $\mathcal{O}(|Q|^2 \cdot |c(Q)|)$. Building it is a straight-forward construction. Once the delayed simulation automaton is built, we have to solve the problem of universal language, i.e. find all states from which all words are accepted. For DBAs this can be done in linear time with depth first search or similar algorithms. □

4.3 Efficiency

Figures 4.1 and 4.2 show that at least on the detspot and detnbaut data sets, some state reduction is achieved. We can see that for all input sizes of automata, reduction via delayed simulation saves up to 50% on detspot and up to 30% on detnbaut, with the average slowly rising as the number of states grows.

Unfortunately, the quadratic run time in the number of states can clearly be seen to have effect in figure 4.3. Already automata with a hundred states can require a run time of over a minute.

4.4 Relation to Moore equivalence

Lemma 4.4.1. $\equiv_M \subseteq \equiv_{de}$

Proof. Let $(\mathcal{A}, p) \equiv_M (\mathcal{B}, q)$ and let $p' = \delta_1^*(p, w)$ and $q' = \delta_2^*(q, w)$ for some w. As $\equiv_M$ is a congruence relation, $(\mathcal{A}, p') \equiv_M (\mathcal{B}, q')$ and therefore $c_1(p') = c_2(q')$. Thus, every run that starts in p' or q' immediately sees a priority of at most $\min\{c_1(p'), c_2(q')\}$. □

Lemma 4.4.2. *For DPAs which are c-normalized and contain no trivial SCCs,* $\equiv_{de} \subseteq \equiv_M$.

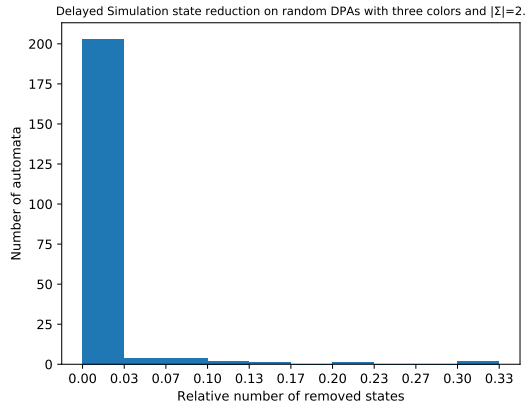
Proof. Assume towards a contradiction that there is a pair $p \equiv_{\text{de}} q$ but $p \not\equiv_M q$. Thus, there is a $w \in \Sigma^*$ with $c(p') \neq c(q')$, where $p' = \delta^*(p, w)$ and $q' = \delta^*(q, w)$. As $\equiv_{\text{de}}$ is a congruence relation, $p' \equiv_{\text{de}} q'$. Without loss of generality, assume $c(p') < c(q')$.

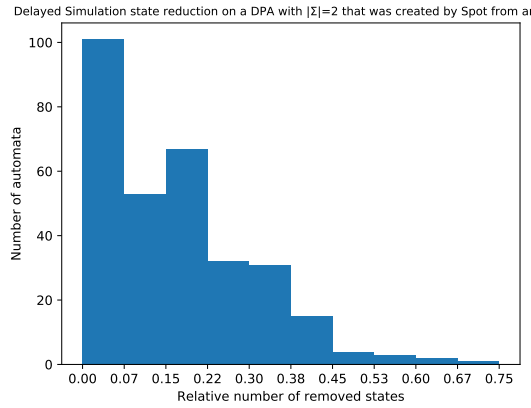
As $\mathcal{A}$ is c-normalized, there is a word u such that $\delta^*(q', u) = q'$ and the lowest priority visited on that path is $c(q')$. Thus, reading u^ω from q' induces a path that never sees a priority of $c(p')$ or lower. This means $p \not\equiv_{\text{de}} q'$. □

4.5 Resetting obligations

In the delayed simulation automaton, "obligations" correspond to good priorities that the first state has accumulated or bad priorities that the second state has accumulated and the need for the respective other state to compensate in some way. The intuitive idea behind this concept is that an obligation that cannot be compensated stands for an infinite run in which the acceptance differs between the two states that are being compared. An issue with the original definition is that obligations carry over forever, even if they can only be caused finitely often. This is demonstrated in figure 4.4; the two states could be merged into one, but they are not $\equiv_{\text{de}}$-equivalent as can be seen in the delayed simulation automaton in figure 4.5.

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





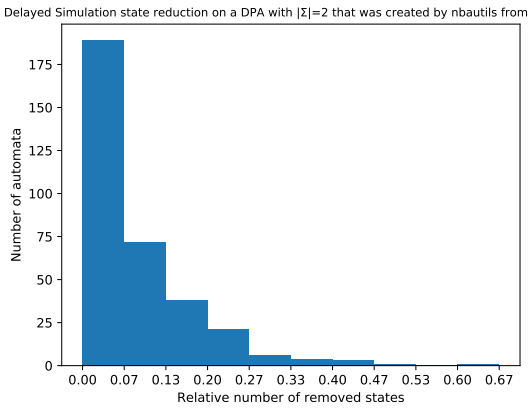
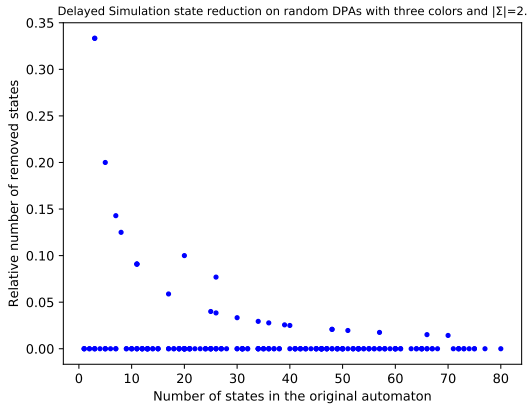
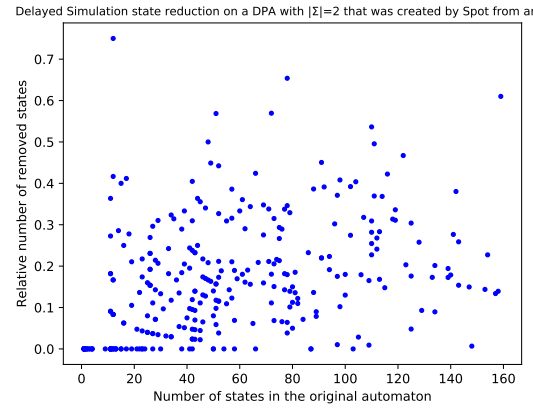


Figure 4.1: State reduction of different automata using μ_{de}.





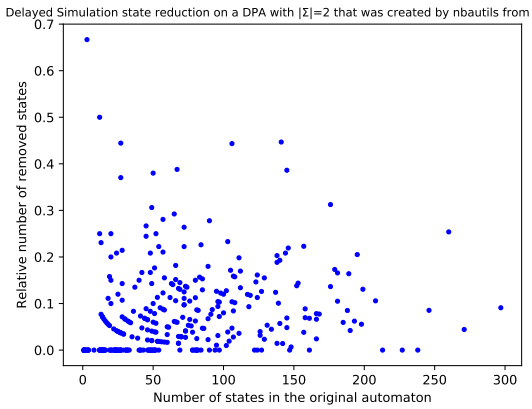


Figure 4.2: Relative state reduction of different automata using μ_{de}.

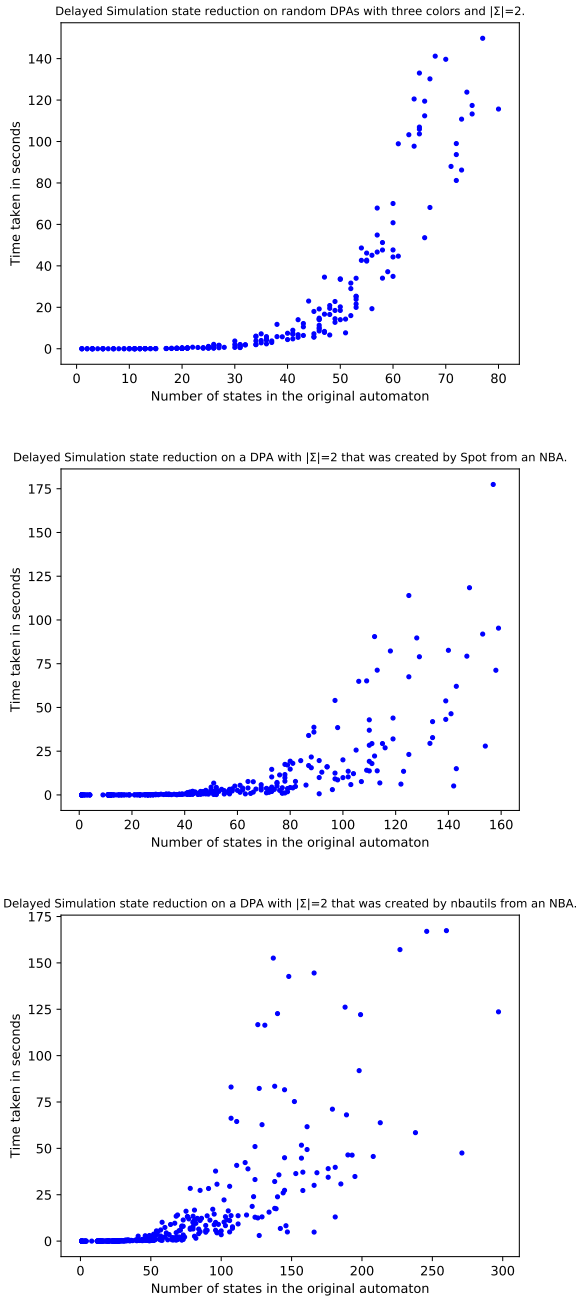


Figure 4.3: Time of the state reduction of different automata using μ_{de}.

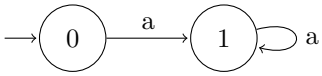


Figure 4.4: Example automaton in which the states could be merged but delayed simulation separates them.

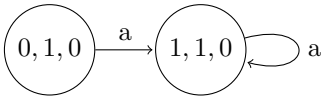


Figure 4.5: Delayed simulation automaton for 4.4.

Definition 4.5.1. Let $\mathcal{A} = (Q, \Sigma, \delta, c)$ be a DPA. We define the *delayed simulation automaton with SCC resets* $\mathcal{A}_{\mathrm{deR}}(p,q) = (Q_{\mathrm{de}}, \Sigma, \delta_{\mathrm{deR}}, F_{\mathrm{de}})$ with $\delta_{\mathrm{deR}}((p,q,k),a) = \delta_{\mathrm{de}}((p,q,\mathrm{reset}(p,q,k,a)),a)$. Except for the addition of the reset function, this automaton is the same as $\mathcal{A}_{\mathrm{de}}$.

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

We write $p \leq_{\mathrm{deR}} q$ if $L(\mathcal{A}_{\mathrm{deR}}(p,q), q_0^{\mathrm{de}}(p,q)) = \Sigma^\omega$. If also $q \leq_{\mathrm{deR}} p$ holds, we write $p \equiv_{\mathrm{deR}} q$.

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

Theorem 4.5.1. *$\equiv_{deR}$ is a congruence relation.*

Lemma 4.5.2. *γ is monotonous in the third component, i.e. if $k \leq_\checkmark k'$, then $\gamma(i,j,k) \leq_\checkmark \gamma(i,j,k')$ for all $i,j \in \mathbb{N}$.*

Proof. We consider each case in the definition of γ. If i is odd, $i \leq_\checkmark k$ and $j \preceq_{\mathrm{p}} i$, then also $i \leq_\checkmark k'$ and $\gamma(i,j,k) = \gamma(i,j,k') = \checkmark$.

If j is even, $j \leq_\checkmark k$ and $j \preceq_{\mathrm{p}} i$, then also $j \leq_\checkmark k'$ and $\gamma(i,j,k) = \gamma(i,j,k') = \checkmark$.

Otherwise, $\gamma(i,j,k) = \min\{i,j,k\}$ and $\gamma(i,j,k') = \min\{i,j,k'\}$. Since $k \leq_\checkmark k'$, $\gamma(i,j,k) \leq_\checkmark \gamma(i,j,k')$. □

Lemma 4.5.3. *$\leq_{de} \subseteq \leq_{deR}$.*

Proof. Consider the two simulation automata $\mathcal{A}_{\mathrm{de}}(p,q)$ and $\mathcal{A}_{\mathrm{deR}}(p,q)$ and let $\alpha \in \Sigma^\omega$ be an arbitrary word. Let $(p_i,q_i,k_i)_{i\in\mathbb{N}}$ and $(p_i,q_i,l_i)_{i\in\mathbb{N}}$ be the runs of these two automata on α. We claim that $k_i \leq_\checkmark l_i$ at every position. Then, since $L(\mathcal{A}_{\mathrm{de}}(p,q), q_0^{\mathrm{de}}(p,q)) = \Sigma^\omega$, both runs must be accepting.

We know $k_0 = l_0$ by definition. For the sake of induction, we look at position $i+1$. If neither p_i nor q_i change their SCC in this step, the statement follows from Lemma 4.5.2. Otherwise, $l_{i+1} = \checkmark \geq_\checkmark k_{i+1}$. □

Unfortunately, Theorem 4.1.4 does not translate easily to $\equiv_{\mathrm{deR}}$. In fact, we were not able to find a characterization for a merger function except for the option to only merge states within single SCCs. Then, however, using the skip merger in combination with normal delayed simulation yields a result just as good.

Chapter 5

Iterated Moore Equivalence

5.1 Theory

Standard Moore equivalence was created to minimize Moore automata in which the priority of every state matters; in other words, the occurrence set of a given run is considered. Parity automata on the other hand only use the smaller infinity set. The idea of the upcoming approach is to utilize this relaxation to give the standard Moore equivalence more freedom and remove additional states.

Definition 5.1.1. Let $\mathcal{A} = (P, \Sigma, \delta, c)$ and $\mathcal{B} = (Q, \Sigma, \varepsilon, d)$ be DPAs and $S \subseteq P$. We say that $\mathcal{A}$ *prepends* S *to* $\mathcal{B}$ if

- $Q \cap S = \emptyset$
- $P = Q \cup S$
- $\delta \restriction_{Q \times \Sigma} = \varepsilon$
- $c \restriction_Q = d$

We assume S to be an SCC in these use cases, i.e. from every $s \in S$, every other $s' \in S$ is reachable in $\mathcal{A}$.

Definition 5.1.2. Let $\mathcal{A} = (Q, \Sigma, \delta, c)$ be a DPA with SCCs $\mathcal{S} \subseteq 2^Q$. Let $\preceq \subseteq \mathcal{S} \times \mathcal{S}$ be a total preorder on the SCCs such that $S \preceq S'$ implies that S' is reachable from S. For the i-th element w.r.t. this order, we write S_i, i.e. $S_0 \prec S_1 \prec \cdots \prec S_{|\mathcal{S}|}$.

For every state q in $\mathcal{A}$, let $\mathrm{SCC}(q)$ be the SCC of q and let $\mathrm{SCCi}(q)$ be the index of that SCC, i.e. $\mathrm{SCC}(q) = S_{\mathrm{SCCi}(q)}$. Let $\preceq_Q \subseteq Q \times Q$ be a total preorder on the states such that $q \preceq_Q q'$ implies $\mathrm{SCCi}(q) \leq \mathrm{SCCi}(q')$.

We inductively define a sequence of automata $(\mathcal{B}_i)_{0 \leq i \leq |\mathcal{S}|}$. For every i, we write $\mathcal{B}_i = (Q_i, \Sigma, \delta_i, c_i)$.

- The state sets are defined as $Q_i = \bigcup_{j=i}^{|\mathcal{S}|} S_j$.
- The base case is $\mathcal{B}_{|\mathcal{S}|} = \mathcal{A} \restriction_{S_{|\mathcal{S}|}}$.
- Given that $\mathcal{B}_{i+1}$ is defined, let $\mathcal{B}'_i = (Q_i, \Sigma, q_0'^i, \delta_i, c'_i)$ be a DPA that prepends S_i to $\mathcal{B}_{i+1}$ such that $\delta \restriction_{Q_i \times \Sigma} = \delta_i$ and $c \restriction_{S_i} = c'_i \restriction_{S_i}$.

- Let M_i' be the Moore equivalence on $\mathcal{B}_i'$. If $S_i = \{q\}$ is a trivial SCC, q is not M_i'-equivalent to any other state, and there is a $p \in Q_{i+1}$ such that for all $a \in \Sigma$, $(\delta_i'(q,a), \delta_i'(p,a)) \in M_i'$, then let p_0 be $\preceq_Q$-maximal among those p and let $c_i(r) = \begin{cases} c_{i+1}(p_0) & \text{if } r = q \\ c_{i+1}(r) & \text{else} \end{cases}$. If any of the three conditions is false, simply set $c_i = c_i'$.

Let M_i be the Moore equivalence on $\mathcal{B}_i$. We define $\equiv_{IM} := M_0$ and call this the *iterated Moore equivalence* of $\mathcal{A}$.

We continuously add the SCCs of $\mathcal{A}$ starting from the "back". In addition to computing the usual Moore equivalence on our automaton, we also take trivial SCCs into special consideration; as their priority cannot appear infinitely often on any run, its value is effectively arbitrary. We can therefore perform extra steps to more liberally merge it with other states.

Definition 5.1.3. Let $\mathcal{A} = (Q, \Sigma, \delta, c)$ be a DPA. We define the *iterated Moore merger function*

$$\mu_{IM} : \mathfrak{C}(\equiv_{IM}, \mathcal{A}) \to 2^Q, \kappa \mapsto \{q \in \kappa \mid q \text{ is } \preceq^{\mathcal{A}}_{\text{reach}} \text{-maximal in } \kappa\}.$$

Theorem 5.1.1. *Let $\mathcal{A}$ be a DPA and let $\mathcal{A}'$ be a representative merge of $\mathcal{A}$ w.r.t. μ_{IM}. For all $q \in Q'$, $(\mathcal{A}, q) \equiv_L (\mathcal{A}', q)$.*

Proof. By definition of the iterated Moore equivalence, we only have $c(p) \neq c'(p)$ if $\{p\}$ is a trivial SCC in $\mathcal{A}$. By having candidates of the merge be $\preceq^{\mathcal{A}}_{\text{reach}}$-maximal, we ensure that for all chosen representatives r_κ, $c(r_\kappa) = c_{|\mathcal{S}|}(r_\kappa)$. With Lemma 2.4.1, this finishes our proof. □

Unfortunately, we were not able to find or prove any relation between iterated Moore equivalence and other relations we established so far. We therefore end this section with two open questions.

- Is $|\mathfrak{C}(\equiv_M, \mathcal{A})| \geq |\mathfrak{C}(\equiv_{IM}, \mathcal{A})|$ true for all $\mathcal{A}$?
- If $\mathcal{A}$ is c-normalized, what is the relation between $\equiv_{IM}$ and $\equiv_{\text{de}}$?

5.2 Computation

Lemma 5.2.1. *For a given DPA $\mathcal{A}$, μ_{IM} can be computed in $\mathcal{O}(|\mathcal{S}| \cdot |Q| \cdot \log |Q|)$.*

Proof. $\equiv_{IM}$ is already defined as an algorithm. We use $|\mathcal{S}|$ many steps of computing Moore equivalence and then checking whether the priority of the newly added state should be changed. The first can be done in log-linear time (Corollary 2.1.12), the second in linear time. This gives us a runtime of $\mathcal{O}(|\mathcal{S}| \cdot |Q| \cdot \log |Q|)$ to calculate $\equiv_{IM}$.

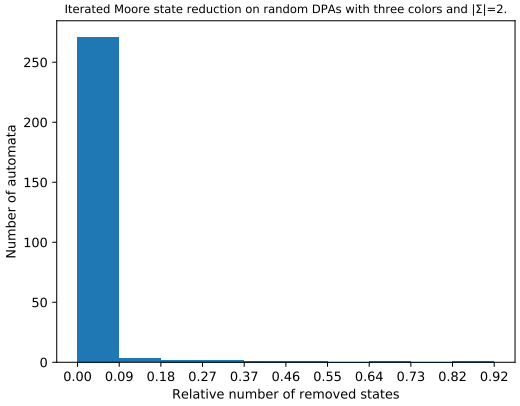
To build μ_{IM} from that can be done in linear time by Lemma 2.3.2. □

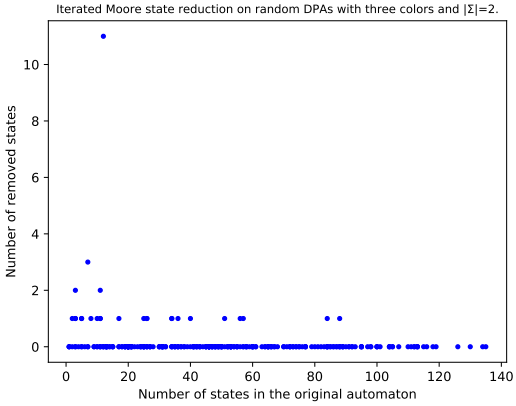
5.3 Efficiency

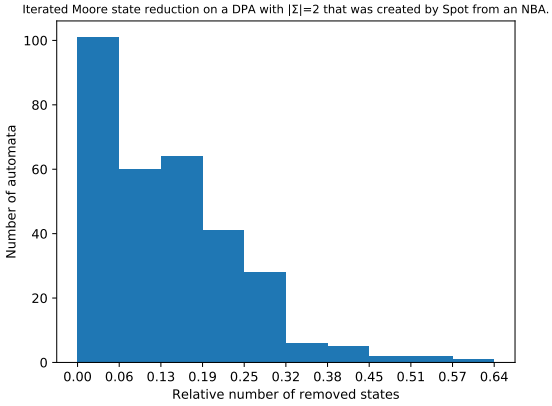
Figure 5.1 shows that the iterated Moore merger actually performs an acceptable job (considering its low complexity) at reducing the number of states in detspot and detnbaut, even when compared to the normal Moore merger in figure 2.3. However, the absolute number of removed states actually stays rather low for detnbaut and consequently the automata that witness a large relative reduction have a small number of states to begin with. This can be seen in figure 5.2: detnbaut rarely has more than 10 states removed, and detspot stays within a similar range of reduction as figure 2.5, with the exception of a few outliers.

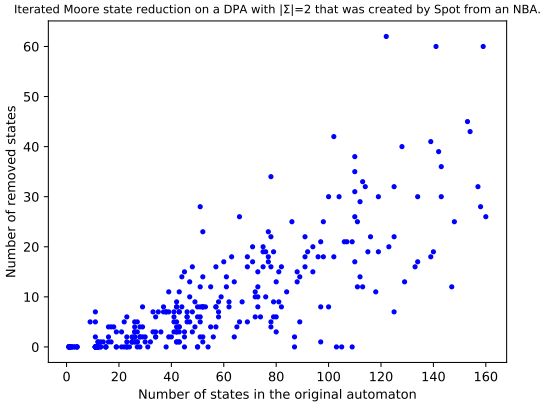
This result is to be expected. As figure 1.3 showed, few automata contain more than three SCCs. The iterated Moore procedure can only exceed the normal Moore quotient by using trivial SCCs.

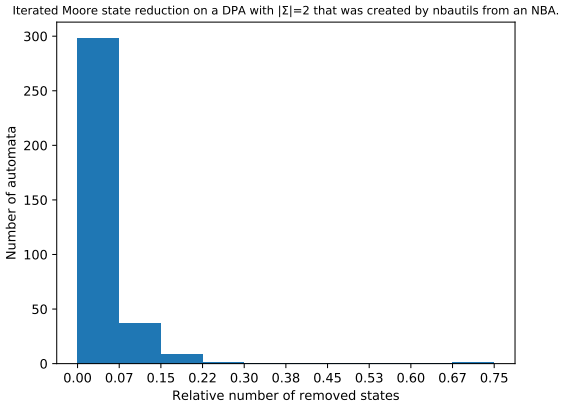
As predicted by the theoretical complexity of the algorithm, figure 5.3 confirms that the iterated Moore merger is fast to apply, only reaching a run time of one second at 200 states.











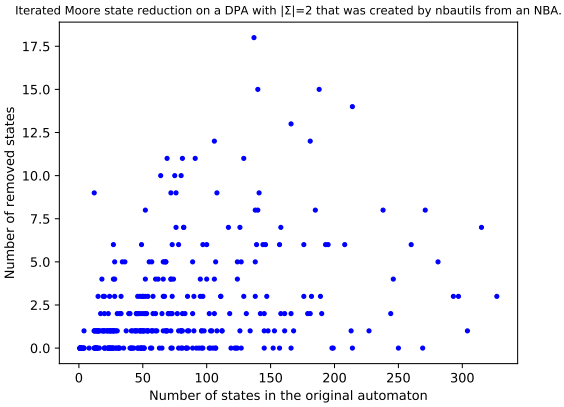


Figure 5.1: State reduction of different automata using μ_{IM}.

Figure 5.2: State reduction of different automata using μ_{IM}.

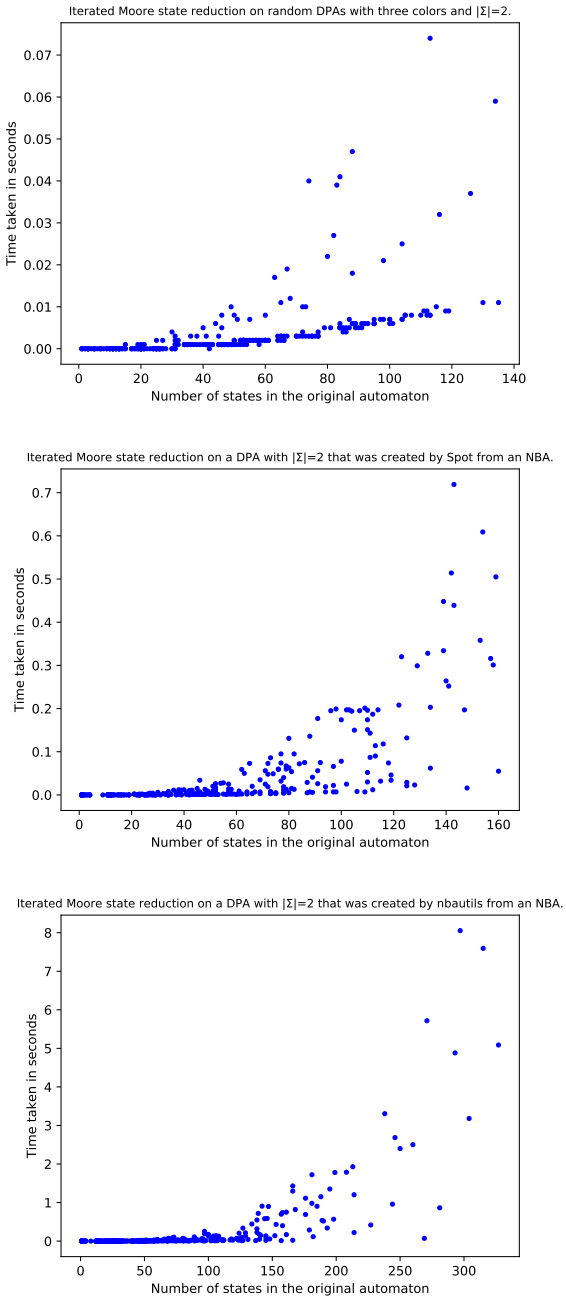


Figure 5.3: Run time of the state reduction of different automata using μ_{IM}.

Chapter 6

Congruence Path Refinement

6.1 Theory

Before we present the path refinement method, note that it takes a special case as it is only defined for state pairs within one DPA, not between different automata.

Definition 6.1.1. Let $\mathcal{A} = (Q, \Sigma, \delta, c)$ be a DPA and let $R \subseteq Q \times Q$ be a congruence relation on the state space. Let $\lambda \subseteq Q$ be an equivalence class of R. We define $L_{\lambda\hookleftarrow}$ as the set of non-empty words w such that for all $q \in \lambda$ and all $u \sqsubseteq w$, $(\delta(q,u),q) \in R$ iff $u \in \{\varepsilon, w\}$. In other words, the set contains all minimal words by which the automaton moves from λ to λ again.

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

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

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

λ being an equivalence class of R, a congruence relation, is required for the definition of $L_{\lambda\hookleftarrow}$ to be consistent.

Lemma 6.1.1. *$\equiv^{\lambda}_{PR}$ is an equivalence relation.*

Proof. The disjoint union of two equivalence relations is an equivalence relation, so we only focus on λ here; it is obvious that $\equiv^{\lambda}_{\mathrm{PR}}$ is an equivalence relation on $Q \setminus \lambda$.

We can actually observe that every X_i is an equivalence relation. For X_0 this is true as that relation is solely defined by equivalence of values. Then by induction it easily follows if X is an equivalence relation that $f_{\mathrm{PR}}(X)$ is one as well. □

Definition 6.1.2. Let $\mathcal{A}$ be a DPA. We define the *path refinement merger* as

$$\mu^{\lambda}_{\mathrm{PR}} : \{\kappa \in \mathfrak{C}(\equiv^{\lambda}_{\mathrm{PR}}, \mathcal{A}) \mid \kappa \subseteq \lambda\} \to 2^{Q}, \kappa \mapsto \{q \in \kappa \mid c(q) = \min c(\kappa)\}.$$

Theorem 6.1.2. *Let $\mathcal{A}'$ be a representative merge of a DPA $\mathcal{A}$ w.r.t μ^{λ}_{PR}. If all states in λ are pairwise language equivalent, then $(\mathcal{A}, q) \equiv_L (\mathcal{A}', q)$ for all $q \in Q'$.*

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

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

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

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

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

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

We abbreviate $\mathrm{Occ}(c(\rho[x, y])) = O[x, y]$ and $\mathrm{Occ}(c(\rho'[x, y])) = O'[x, y]$. c and c' map all states to the same values by definition of the merge, so $c'(\rho'(i))$ and $c(\rho'(i))$ are the same.

First, observe that for all i, $\min O[k_i, k_{i+1} + 1] \geq \min O'[k_i, k_{i+1} + 1]$: By definition of $\equiv^{\lambda}_{\mathrm{PR}}$, all states within an equivalence class visit the same minimal priority on their way from λ back to λ (in $\mathcal{A}$). In particular, $\rho(k_i)$ and $\rho'(k_i)$ do. That means $\min O[k_i, k_{i+1} + 1] = \min O'[k_i, k_{i+1}] \cup c(p_{i+1})$, where $p_{i+1} = \delta(\rho'(k_{i+1}), \alpha(k_{i+1}))$. By choice of the candidates of the merge, $c(p_{i+1}) \geq c(\rho'(k_{i+1}))$, so $\min O'[k_i, k_{i+1}] \cup c(p_{i+1}) \geq \min O'[k_i, k_{i+1}+1]$. We can extend this to $\min \mathrm{Inf}(c(\rho)) \geq \min \mathrm{Inf}(c(\rho'))$.

Now assume that there is a priority l in $c(\rho')$ that occurs infinitely often but only finitely often in $c(\rho)$. Let this priority be minimal and let k_m be a position with $c(\rho'(k_m)) = l$ but l never occurs in $c(\rho)$ at or after position k_m.

We still have $\rho(k_m) \equiv^{\lambda}_{\mathrm{PR}} \rho'(k_m)$, so $\min O[k_m, k_{m+1}+1]$ and $\min O'[k_m, k_{m+1}] \cup \{c(p_{m+1})\}$ must be the same. Obviously, l is an element of $O'[k_m, k_{m+1}]$, so a priority at most as high as l is visited in $\rho[k_m, k_{m+1} + 1]$. This happens every time l is seen in $c(\rho')$, so $l > \min \mathrm{Inf}(c(\rho'))$. As l was the smallest priority that differs between the two infinity sets, their minimum must be the same. □

6.2 Computation

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

Definition 6.2.1. Let $\mathcal{A} = (Q, \Sigma, \delta, c)$ be a DPA and let $R \subseteq Q \times Q$ be a congruence relation. For each equivalence class λ of R, we define the *path refinement automaton* $\mathcal{G}^{\lambda}_{\mathrm{PR}}(p, q) = (Q_{\mathrm{PR}}, \Sigma, \delta^{\lambda}_{\mathrm{PR}}, F_{\mathrm{PR}})$, which is a DFA.

- $Q_{\mathrm{PR}} = (Q \times Q \times c(Q) \times \{<, >, =\}) \cup \{\bot\}$

- $\delta^{\lambda}_{\mathrm{PR}}((p,q,k,x),a) = \begin{cases} (p',q',\eta_k(c(p'),c(q'),k),\eta_x(c(p'),c(q'),k,x)) & \text{if } p' \notin \lambda \\ q_0^{\mathrm{PR}}(p',q') & \text{if } p' \in \lambda \text{ and} \\ & \quad (\eta_x(c(p'),c(q'),k,x) ==) \\ \bot & \text{else} \end{cases}$

  where $p' = \delta(p,a)$ and $q' = \delta(q,a)$.

  $\eta_k(k_p,k_q,k) = \min_{\leq_\checkmark}\{k_p,k_q,k\}$

  $\eta_x(k_p,k_q,k,x) = \begin{cases} < & \text{if } (k_p <_\checkmark k_q \text{ and } k_p <_\checkmark k) \text{ or } (k < k_q \text{ and } (x = <)) \\ > & \text{if } (k_p >_\checkmark k_q \text{ and } k >_\checkmark k_q) \text{ or } (k_p > k \text{ and } (x = >)) \\ = & \text{else} \end{cases}$

- $F_{\mathrm{PR}} = Q_{\mathrm{PR}} \setminus \{\bot\}$

We use the following initial state for $p,q \in \lambda$:

$$q_0^{\mathrm{PR}}(p,q) = (p,q,\eta_k(c(p),c(q),\checkmark),\eta_x(c(p),c(q),\checkmark,=))$$

Lemma 6.2.1. *Let p and q be two states in λ and let w be a word in $L_{\lambda\to\lambda}$. For every $v \sqsubset w$ and $\oplus \in \{<,>,=\}$, the fourth component of $(\delta^{\lambda}_{PR})^*(q_0^{PR}(p,q),v)$ is $\oplus$ if and only if $\min\{c(\delta^*(p,u)) \mid u \sqsubseteq v\} \oplus \min\{c(\delta^*(q,u)) \mid u \sqsubseteq v\}$.*

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

Theorem 6.2.2. *Let $p,q \in \lambda$. Then $p \equiv^{\lambda}_{PR} q$ iff $L(\mathcal{G}^{\lambda}_{PR}(p,q), q_0^{PR}(p,q)) = \Sigma^*$.*

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

If $m = 0$, then $(p,q) \notin Y_\lambda$, meaning that there is a word w such that $\min\{c(\delta^*(p,u)) \mid u \sqsubset w\} \neq \min\{c(\delta^*(q,u)) \mid u \sqsubset w\}$. Without loss of generality, assume $\min\{c(\delta^*(p,u)) \mid u \sqsubset w\} < \min\{c(\delta^*(q,u)) \mid u \sqsubset w\}$. By Lemma 6.2.1, $x_{|w|-1} = <$. Furthermore, $\delta(p_{|w|-1}, w_{|w|-1}) \in \lambda$, as $w \in L_{\lambda\to\lambda}$. Thus, $\rho(|w|) = \bot$ and the run is rejecting.

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

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

If $m = 1$, then $w \in L_{\lambda\to\lambda}$. Since $\rho(|w|) = \bot$, it must be true that $x_{|w|-1} \neq =$. Without loss of generality, assume $x_{|w|-1} = <$. By Lemma 6.2.1, $\min\{c(\delta^*(p,u)) \mid u \sqsubseteq w\} < \min\{c(\delta^*(q,u)) \mid u \sqsubseteq w\}$. Therefore, $p \not\equiv^{R}_{\mathrm{PR}} q$.

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

The automaton $\mathcal{G}^{\lambda}_{\mathrm{PR}}$ has size $\mathcal{O}(|Q|^2 \cdot |c(Q)|)$. For each class λ, the problem of universal language has to be solved which is solved by reachability in linear time. Thus, $\equiv^{\lambda}_{\mathrm{PR}}$ can be computed in time $\mathcal{O}(|Q|^2 \cdot |c(Q)|)$ but has to be repeated for every different λ.

6.3 Faster Computation

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

Definition 6.3.1. Let $\mathcal{A} = (Q, \Sigma, \delta, c)$ be a DPA. Let R be a congruence relation on Q and let $\lambda \subseteq Q$ be an equivalence class of R. We define a deterministic transition structure $\mathcal{A}^{\lambda}_{\mathrm{visit}} = (Q^{\lambda}_{\mathrm{visit}}, \Sigma, \delta^{\lambda}_{\mathrm{visit}})$ as follows:

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

Definition 6.3.2. Consider $\mathcal{A}^{\lambda}_{\mathrm{visit}}$ of a DPA $\mathcal{A}$ and a congruence relation R. We define an equivalence relation $V \subseteq Q^{\lambda}_{\mathrm{visit}} \times Q^{\lambda}_{\mathrm{visit}}$ as:

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

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

Lemma 6.3.1. *For all $p, q \in \lambda$ and $l, k \in c(Q)$: $(\iota^l_p, \iota^k_q) \in V$ iff $l = k$.*

Proof. Follows directly from the definition. □

Lemma 6.3.2. *Let $q \in \lambda$, $k \in c(Q)$, $w \in L_{\lambda\hookleftarrow}$, and $\varepsilon \sqsubset v \sqsubset w$. Then $(\delta^{\lambda}_{visit})^*(\iota^k_q, v) = (\delta^*(q,v), x_v, \bot)$, where $x_v = \min\{c(\delta^*(q,u)) \mid u \sqsubseteq v\}$.*

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

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

Lemma 6.3.3. *Let $q \in \lambda$, $k \in c(Q)$, and $w \in L_{\lambda\hookleftarrow}$. Then $(\delta^{\lambda}_{visit})^*(\iota^k_q, w) = \iota^x_{q'}$, where $q' = \delta^*(q,w)$ and $x = \min\{c(\delta^*(q,u)) \mid u \sqsubseteq w\}$.*

Proof. For all $v \sqsubseteq w$, let $x_v = \min\{c(\delta^*(q,u)) \mid u \sqsubseteq v\}$ (i.e. $x = x_w$). Let $w = va \in \Sigma^* \cdot \Sigma$ (since words in $L_{\lambda\hookleftarrow}$ are non-empty). Then $(\delta^\lambda_{\text{visit}})^*(\iota^k_q, v) = (\delta^*(q,v), x_v, \bot)$ by Lemma 6.3.2 and $(\delta^\lambda_{\text{visit}})^*(\iota^k_q, w) = \delta^\lambda_{\text{visit}}((\delta^*(q,v), x_v, \bot), a)$.

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

Lemma 6.3.4. *For every $q \in \lambda$, $l, k \in c(Q)$, and $w \in \Sigma^+$: $(\delta^\lambda_{visit})^*(\iota^k_q, w) = (\delta^\lambda_{visit})^*(\iota^l_q, w)$.*

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

For $w \in \Sigma$, w is always in $L_{\lambda\hookleftarrow}$ or a prefix of a word in that set. Thus we can apply Lemma 6.3.2 and 6.3.3 to obtain our wanted result. □

Lemma 6.3.5. *For all $p, q \in \lambda$, and $l, k \in c(Q)$, $(\iota^k_p, \iota^k_q) \in V_M$ if and only if $(\iota^l_p, \iota^l_q) \in V_M$.*

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

It must be true that $w \neq \varepsilon$; otherwise, $(\iota^l_p, \iota^l_q) \notin V$, which would contradict Lemma 6.3.1.

By Lemma 6.3.4, we have $(\delta^\lambda_{\text{visit}})^*(\iota^l_p, w) = (\delta^\lambda_{\text{visit}})^*(\iota^k_p, w)$ and analogously for q. Therefore, $((\delta^\lambda_{\text{visit}})^*(\iota^k_p, w), (\delta^\lambda_{\text{visit}})^*(\iota^k_q, w)) \notin V$ and thus $(\iota^k_p, \iota^k_q) \notin V_M$, which contradicts our assumption. □

Lemma 6.3.6. *Let $q \in \lambda$ and $w \in \Sigma^+$ such that $\delta^*(q,w) \in \lambda$. Then there is a decomposition of w into words $v_1 \cdots v_m$ such that all $v_i \in L_{\lambda\hookleftarrow}$.*

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

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

Theorem 6.3.7. *Let $\hat{c} = \max c(Q)$. Then for all $p, q \in \lambda$, we have $p \equiv^\lambda_{PR} q$ iff $(\iota^{\hat{c}}_p, \iota^{\hat{c}}_q) \in V_M$.*

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

If Assume there is a w such that $((\delta^\lambda_{\text{visit}})^*(\iota^{\hat{c}}_p, w), (\delta^\lambda_{\text{visit}})^*(\iota^{\hat{c}}_q, w)) \notin V$. Choose this w to have minimal length. By definition of V, both $(\delta^\lambda_{\text{visit}})^*(\iota^{\hat{c}}_p, w)$ and $(\delta^\lambda_{\text{visit}})^*(\iota^{\hat{c}}_q, w)$ must be in λ. By Lemma 6.3.6, there is a decomposition of w into words $v_1 \cdots v_m$ that are in $L_{\lambda\hookleftarrow}$. We perform a proof of induction on m.

If $m = 1$, then $w \in L_{\lambda\hookleftarrow}$. From Lemmas 6.3.1 and 6.3.3, we know that $\min\{c(\delta^*(p,u)) \mid u \sqsubseteq w\} \neq \min\{c(\delta^*(q,u)) \mid u \sqsubseteq w\}$ and therefore $p \not\equiv^\lambda_{\text{PR}} q$.

If $m + 1 > 1$, consider $(\delta^\lambda_{\text{visit}})^*(\iota^{\hat{c}}_q, v_1) = \iota^x_{q'}$ and $(\delta^\lambda_{\text{visit}})^*(\iota^{\hat{c}}_q, v_1) = \iota^y_{p'}$ as stated in Lemma 6.3.3 (with $p' = \delta^*(p, v_1)$ and q' analogously). w was chosen to have minimal length, so $(\iota^x_{q'}, \iota^y_{p'}) \in V$, which means that $x = y$.

As $(\iota^{\hat{c}}_p, \iota^{\hat{c}}_q) \notin V_M$, we also have $(\iota^x_{p'}, \iota^y_{q'}) \notin V_M$ and by Lemma 6.3.5, $(\iota^{\hat{c}}_{p'}, \iota^{\hat{c}}_{q'}) \notin V_M$ with the word $v_2 \cdots v_m$ being a witness. We can therefore argue with induction to deduce $p' = \delta^*(p, v_1) \not\equiv^\lambda_{\text{PR}} q' = \delta^*(q, v_1)$. As $v_1 \in L_{\lambda\hookleftarrow}$, the definition of path refinement tells us $p \not\equiv^\lambda_{\text{PR}} q$.

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

If $n = 0$, there is a word $w \in L_{\lambda\hookleftarrow}$ such that $\min\{c(\delta^*(p,u)) \mid u \sqsubseteq w\} \neq \min\{c(\delta^*(q,u)) \mid u \sqsubseteq w\}$. Let $x, y \in c(Q)$ be the third components of $(\delta^{\lambda}_{\mathrm{visit}})^*(\iota^{\hat{c}}_p, w)$ and $(\delta^{\lambda}_{\mathrm{visit}})^*(\iota^{\hat{c}}_q, w)$ respectively. By Lemma 6.3.3, $x = \min\{c(\delta^*(p,u)) \mid u \sqsubseteq w\}$ and $y = \min\{c(\delta^*(q,u)) \mid u \sqsubseteq w\}$, so $x \neq y$. By definition of $\mathcal{A}^{\lambda}_{\mathrm{visit}}$, this means that $((\delta^{\lambda}_{\mathrm{visit}})^*(\iota^{\hat{c}}_p, w), (\delta^{\lambda}_{\mathrm{visit}})^*(\iota^{\hat{c}}_q, w)) \notin V$.

For $n+1 > 0$, there is a $w \in L_{\lambda\hookleftarrow}$ such that $(\delta^*(p,w), \delta^*(q,w)) \notin V$. Let $p' = \delta^*(p,w)$ and q' analogously. By induction, $(\iota^{\hat{c}}_{p'}, \iota^{\hat{c}}_{q'}) \notin V_M$. Lemma 6.3.3 tells us that there are k' and l' such that $(\delta^{\lambda}_{\mathrm{visit}})^*(\iota^{\hat{c}}_p, w) = \iota^{k'}_{p'}$ and $(\delta^{\lambda}_{\mathrm{visit}})^*(\iota^{\hat{c}}_q, w) = \iota^{l'}_{q'}$. Since n was chosen to be minimal, it must be true that $k' = l'$; otherwise, we would already have $p, q \notin X_0$. From Lemma 6.3.5, we know that $(\iota^{\hat{c}}_{p'}, \iota^{\hat{c}}_{q'}) \notin V_M$ if and only if $(\iota^{k'}_{p'}, \iota^{l'}_{q'}) \notin V_M$, which is false. Thus, finally, $(\iota^{\hat{c}}_p, \iota^{\hat{c}}_q) \notin V_M$. □

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

6.4 Efficiency

For the analysis of efficiency, we performed the merge w.r.t. $\mu^{\lambda}_{\mathrm{PR}}$ for all $\lambda \in \mathfrak{C}(R, \mathcal{A})$ in succession. In general, we used $R = \equiv_L$. As usual, figures 6.1 and 6.2 show that gendet is not susceptible for state space reduction. The path refinement merger shows great promise on detspot and detnbaut though, frequently achieving reduction rates of over 20%.

Figure 6.3 displays the run time required for the entire reduction procedure. We can see that it is one of the slower algorithms, similar to delayed simulation. This is, however, not due to the path refinement itself but rather due to the computation of $\equiv_L$ which we use for its equivalence classes.

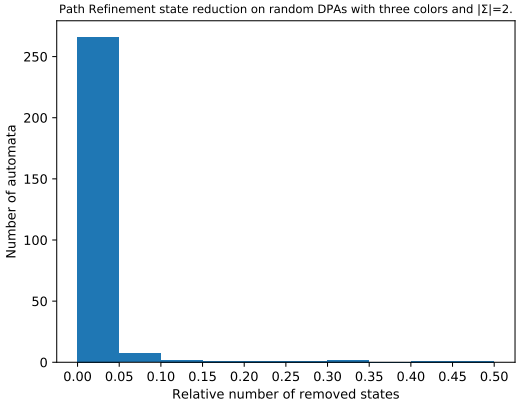
The generation of detnbaut also implicitly outputs a relation that implies language equivalence which has to be computed for the determination regardless. We can therefore use that relation without any additional cost. As can be seen in figure 6.4 and 6.5, using this relation still leads to a relative reduction of about 5 to 15%. The run time graph also more closely resembles a linear function and is only a fraction of what is required to compute $\equiv_L$ (figure 6.6).

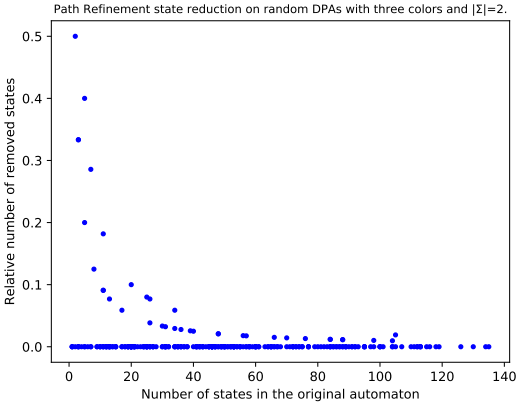
6.5 Multiple Merges

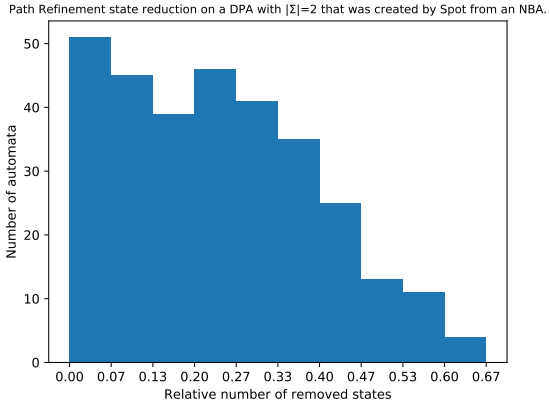
So far, we always considered $\mu^{\lambda}_{\mathrm{PR}}$ for some fixed equivalence class λ. However, building this merger only touches states in λ; all other states of the automaton remain as before.

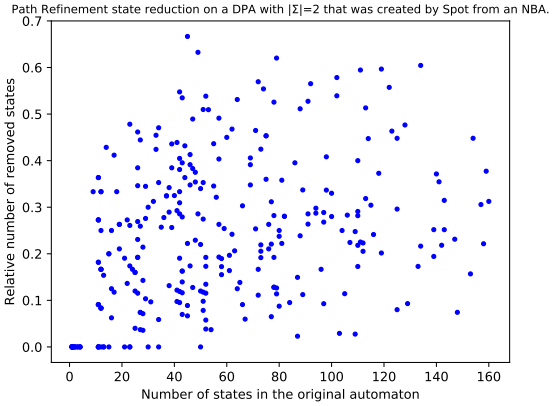
From Theorem 6.1.2, we can deduce that if R is a congruence relation that implies language equivalence on $\mathcal{A}$, then $R \cap (Q' \times Q')$ is the same on a representative merge $\mathcal{A}'$ (w.r.t. $\mu^{\lambda}_{\mathrm{PR}}$). This tells us that we can simply "reuse" the relation R, or rather its equivalence classes, to compute the path refinement relation for different λ and sequentially built their merges. The upcoming example shows that this can provide further state reduction and more importantly that the order that the equivalence classes are considered in matters.

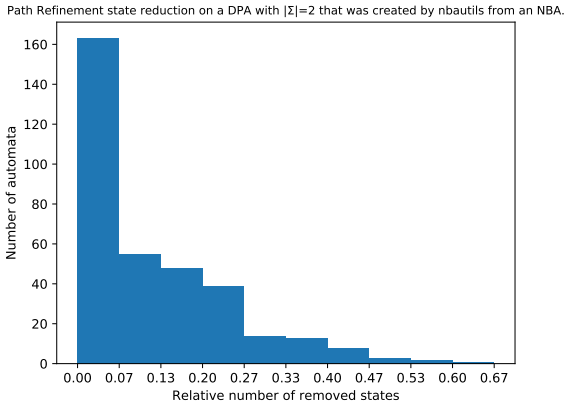
The automaton displayed in figure 6.7 has six states. Assume our relation has four classes, $\{q_1, q_5\}$, $\{q_2, q_3\}$, $\{q_4\}$, and $\{q_0\}$. Equivalence classes of size 1 cannot cause any state reduction, so we focus on $\lambda_1 = \{q_1, q_5\}$ and $\lambda_2 = \{q_2, q_3\}$.











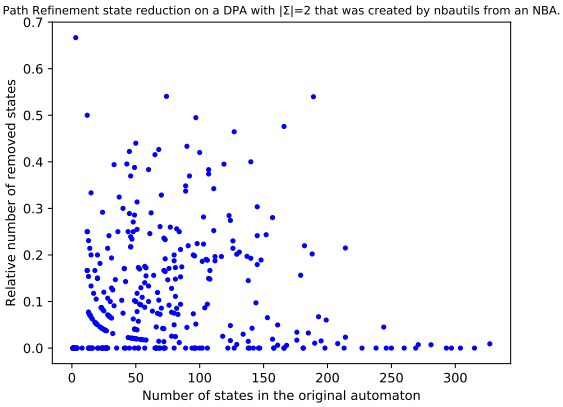


Figure 6.1: State reduction of different automata using $\mu^{\lambda}_{\mathrm{PR}}$.

Figure 6.2: Relative state reduction of different automata using $\mu^{\lambda}_{\mathrm{PR}}$.

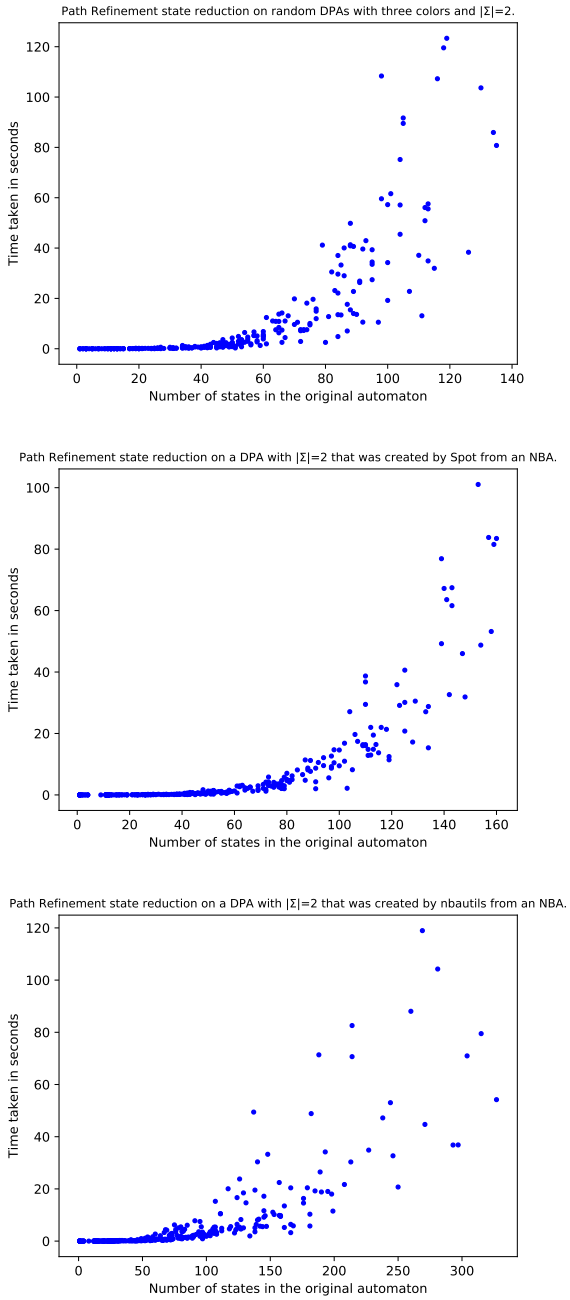


Figure 6.3: Time of the state reduction of different automata using $\mu^{\lambda}_{\mathrm{PR}}$.

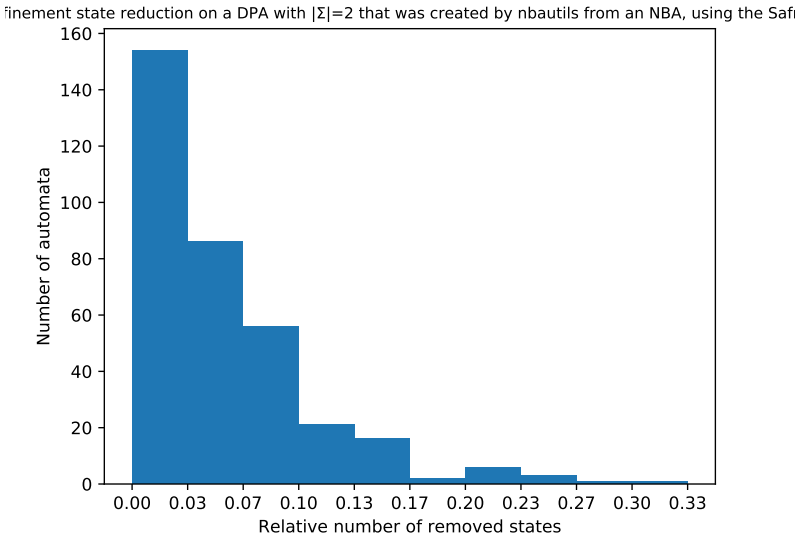


Figure 6.4: State reduction of different automata using $\mu^{\lambda}_{\mathrm{PR}}$.

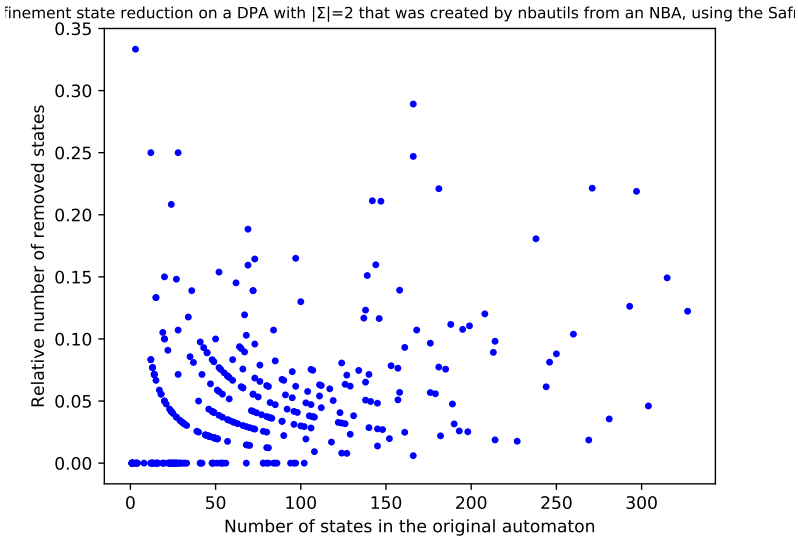


Figure 6.5: State reduction of different automata using $\mu^{\lambda}_{\mathrm{PR}}$.

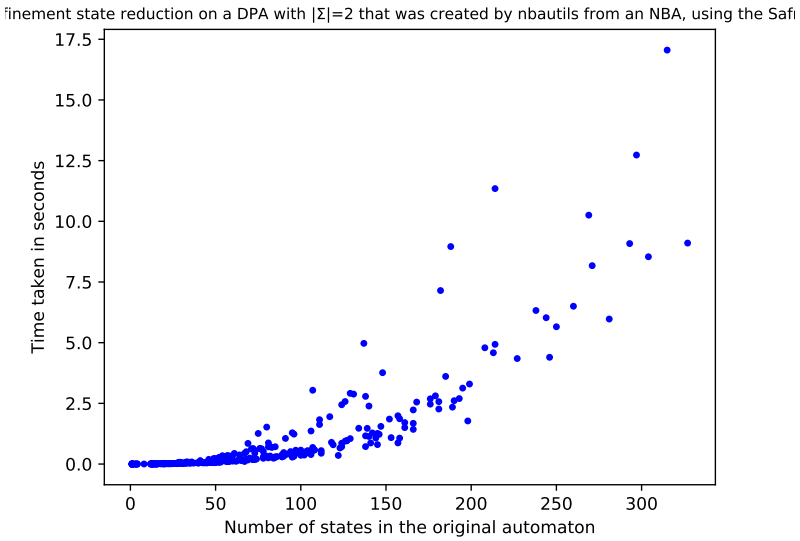


Figure 6.6: State reduction of different automata using $\mu_{\mathrm{PR}}^{\lambda}$.

Consider $\mathcal{A}_1$, a representative merge w.r.t. $\mu_{\mathrm{PR}}^{\lambda_1}$. From q_1, there is a path back to λ_1 again that only visits priority 1, namely $q_1q_3q_4q_5$. That is impossible from q_5, as the first step always leads to q_0 with priority 0. Hence, no merge will occur in this step.

On the other hand consider $\mathcal{A}_2$, a representative merge w.r.t. $\mu_{\mathrm{PR}}^{\lambda_2}$. From the pair (q_2, q_3), every path back to λ_2 will either move through q_2 or q_0 and thus have the minimal priority 0. The resulting automaton is shown in figure 6.8.

Now look at $\mathcal{A}_{21}$, a representative merge of $\mathcal{A}_2$ w.r.t. $\mu_{\mathrm{PR}}^{\lambda_1}$. As q_3 does not exist anymore, the path $q_1q_3q_4q_5$ becomes $q_1q_2q_4q_5$ with minimal priority of 0. In fact, now q_1 and q_5 can be merged, unlike before.

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

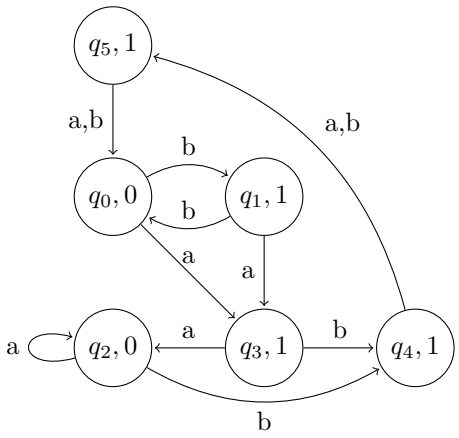


Figure 6.7: Example automaton for which the order of congruence classes matters in path refinement.

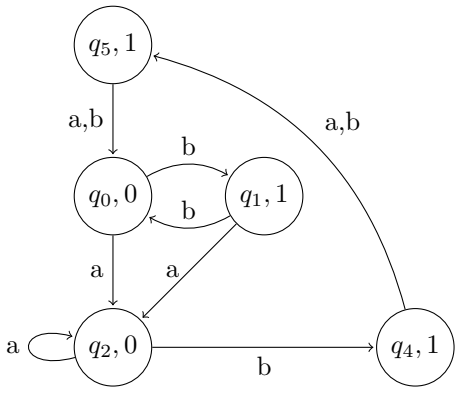


Figure 6.8: Automaton from figure 6.7 after merging q_2 and q_3.

Chapter 7

Threshold Moore

7.1 Theory

Definition 7.1.1. 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 $c'(q) = \begin{cases} c(q) & \text{if } c(q) \leq k \\ k+1 & \text{else} \end{cases}$.

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 7.1.1. *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. This follows directly by plugging in definitions. For example, assume that $\approx = \mathcal{T}(\sim, > k)$ is not symmetric, so $(\mathcal{A}, p) \approx (\mathcal{B}, q)$ but $(\mathcal{B}, q) \not\approx (\mathcal{A}, p)$. Then $(\mathcal{T}(\mathcal{A}, > k), p) \sim (\mathcal{T}(\mathcal{B}, > k), q)$ but $(\mathcal{T}(\mathcal{B}, > k), q) \not\sim (\mathcal{T}(\mathcal{A}, > k), p)$ and thus $\sim$ is not symmetric either. □

Definition 7.1.2. We set $\equiv_M^{\leq k} := \mathcal{T}(\equiv_M, > k)$.

Lemma 7.1.2. *For two numbers $x, y \in \mathbb{N}$, let $x =^{\leq k} y$ iff $x = y$ or $\min\{x, y\} > k$. Then $(\mathcal{A}, p) \equiv_M^{\leq k} (\mathcal{B}, q)$ iff for all $w \in \Sigma^*$, $c_1(\delta_1^*(p, w)) =^{\leq k} c_2(\delta_2^*(q, w))$.*

Proof. $(\mathcal{A}, p) \equiv_M^{\leq k} (\mathcal{B}, q)$ is true iff $(\mathcal{T}(\mathcal{A}, > k), p) \equiv_M (\mathcal{T}(\mathcal{B}, > k), q)$ iff for all $w \in \Sigma^*$, $c_1'(\delta_1^*(p, w)) = c_2'(\delta_2^*(q, w))$. If we compare definitions, it becomes clear that $c_1'(x) = c_2'(y)$ iff $c_1(x) =^{\leq k} c_2(y)$. □

Definition 7.1.3. 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 $\equiv^{\sim}_{\mathrm{TM}}$ 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 $\kappa \in \mathfrak{C}(\equiv^{\sim}_{\mathrm{TM}})$ be a set of states in $\mathcal{A}$. By definition of $\equiv^{\sim}_{\mathrm{TM}}$, 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 7.1.3. *For all $q \in Q'$, $(\mathcal{A}, q) \equiv_L (\mathcal{A}', q)$.*

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

By Lemma 2.2.3, $(\mathcal{A}, \rho(i)) \equiv_L (\mathcal{A}, \rho'(i))$ and $(\mathcal{A}, \rho(i)) \equiv^{\leq k}_M (\mathcal{A}, \rho'(i))$ 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. □

The research becomes more interesting if you fix $\sim$ to be the complete language equivalence $\equiv_L$, which is what we consider for the next Lemmas.

Lemma 7.1.4. *For all $q \in Q'$ with $c(q) \leq k$, $(\mathcal{A}, q) \equiv^{\leq k}_M (\mathcal{A}', q)$.*

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 $c(\rho(i)) =^{\leq k} c'(\rho'(i))$ for all i which then proves the Lemma.

By Lemma 2.2.3, $(\mathcal{A}, \rho(i)) \equiv^{\leq k}_M (\mathcal{A}, \rho'(i))$ 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 $c(\rho(i)) =^{\leq k} c'(\rho'(i))$. □

Lemma 7.1.5. *Let $l \leq k \in \mathbb{N}$. Then $\equiv^{\leq k}_M \subseteq \equiv^{\leq l}_M$.*

Proof. Let $\mathcal{A} = (Q_1, \Sigma, \delta_1, c_1)$ and $\mathcal{B} = (Q_2, \Sigma, \delta_2, c_2)$ be two DPAs. Let $\mathcal{A}^{\leq l} = \mathcal{T}(\mathcal{A}, > l)$ and $\mathcal{A}^{\leq k}$, $\mathcal{B}^{\leq l}$, and $\mathcal{B}^{\leq k}$ analogously.

Let $(\mathcal{A}, p) \not\equiv^{\leq l}_M (\mathcal{B}, q)$, so there is a word $w \in \Sigma^*$ such that $c_1^{\leq l}(p') \neq c_2^{\leq l}(q')$, where $p' = \delta_1^*(p, w)$ and $q' = \delta_2^*(q, w)$. This means at least one of $c_1(p')$ and $c_2(q')$ is at most l, as otherwise both these values would be set to $l + 1$. Due to symmetry, we can assume that $c_1(p')$ is that value.

As $l \leq k$, $c_1^{\leq k}(p') = c_1^{\leq l}(p')$ and $c_2^{\leq k}(q') \geq c_2^{\leq l}(q')$, so $c_1^{\leq k}(p') \neq c_2^{\leq k}(q')$ and thus $(\mathcal{A}, p) \not\equiv^{\leq k}_M (\mathcal{B}, q)$. □

Lemma 7.1.6. *For all $q \in Q'$ with $c(q) \leq k$, $(\mathcal{A}, q) \equiv^{\equiv_L}_{TM} (\mathcal{A}', q)$.*

Proof. Representative merges never change priorities assigned to states. Together with Lemma 7.1.3, Lemma 7.1.4, and Lemma 7.1.5 this already finishes the proof. □

Lemma 7.1.7. *For all $p, q \in Q'$ with $\min\{c(p), c(q)\} \leq k$, $(\mathcal{A}, p) \equiv^{\equiv_L}_{TM} (\mathcal{A}, q)$ iff $(\mathcal{A}', p) \equiv^{\equiv_L}_{TM} (\mathcal{A}', q)$.*

Proof. If $c(p) = c(q) \leq k$: By Lemma 7.1.6, we know that $(\mathcal{A}, p) \equiv^{\equiv_L}_{\mathrm{TM}} (\mathcal{A}', p)$ and $(\mathcal{A}, q) \equiv^{\equiv_L}_{\mathrm{TM}} (\mathcal{A}', q)$. If now $(\mathcal{A}, p) \equiv^{\equiv_L}_{\mathrm{TM}} (\mathcal{A}, q)$ holds, then also $(\mathcal{A}', p) \equiv^{\equiv_L}_{\mathrm{TM}} (\mathcal{A}', q)$ and vice versa.

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

Definition 7.1.4. 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 7.1.8. *Every representative merge of a DPA $\mathcal{A}$ w.r.t. $\mu_{TM}^{\equiv_L}$ is language equivalent to $\mathcal{A}$.*

Proof. Building a representative merge of $\mathcal{A}$ w.r.t. $\mu_{\mathrm{TM}}^{\equiv_L}$ implicitly builds multiple representative merges $\mathcal{A}_0, \ldots, \mathcal{A}_m$, where $\mathcal{A}_{i+1}$ is a representative merge of $\mathcal{A}_i$ w.r.t. some κ_i. By Lemma 7.1.7, for all $j > i$, the states in κ_j are still pairwise $\equiv_{\mathrm{TM}}^{\equiv_L}$ equivalent. Moreover, κ_{i+1} is an $\equiv_{\mathrm{TM}}^{\equiv_L}$ equivalence class in $\mathcal{A}_i$. 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 7.1.3. □

7.2 Computation

Lemma 7.2.1. *For a given DPA, $\equiv_M^{\leq k}$ can be computed in $\mathcal{O}(|Q| \cdot \log |Q|)$.*

Proof. Computing $\mathcal{T}(\mathcal{A}, > k)$ can clearly be done in linear time. The rest follows from Corollary 2.1.12. □

Lemma 7.2.2. *For a given $\mathcal{A}$ and equivalence relation $\sim$, $\mu_{TM}^{\sim}$ can be computed in $\mathcal{O}(|Q| \cdot \log |Q| \cdot |c(Q)|)$.*

Proof. First, we compute $\equiv_M^{\leq k}$ for all $k \in c(Q)$. This can be done in $\mathcal{O}(|Q| \cdot \log |Q| \cdot |c(Q)|)$ by Lemma 7.2.1. After that, we build the restriction of $\equiv_M^{\leq k}$ to $c^{-1}(k)$ for every k and then again find the union of all those sets. Both steps can be done in linear time. Finally, the result of this operation has to be intersected with $\sim$ which is also possible in linear time given a suitable data structure for $\sim$. This then gives us $\equiv_{\mathrm{TM}}^{\sim}$. As $\mu_{\mathrm{TM}}^{\sim}$ is a simple quotient merger, there is no additional work to be done. □

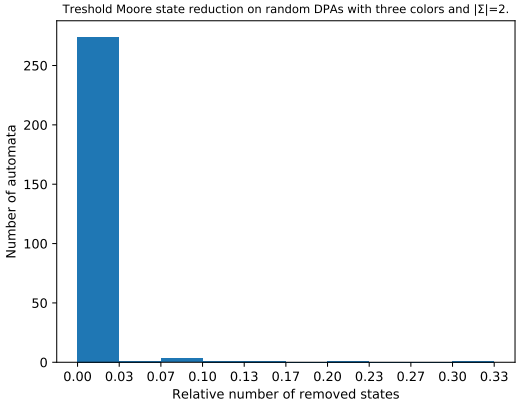
7.3 Efficiency

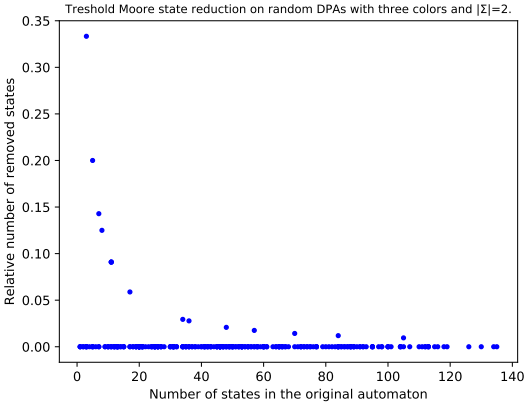
Figures 7.1 and 7.2 show that, unfortunately, the Threshold Moore merger only has very little effect on the gendet and detnbaut data sets. Comparing the results for detspot to those of μ_M in figure 2.3 shows that there is not much gain here either. Even though the results of the reduction are disappointing, it still requires us to compute some kind of relation $\sim$; in our case we used $\equiv_L$ again, which also had a noticeable impact on the run time as can be seen in figure 7.3.

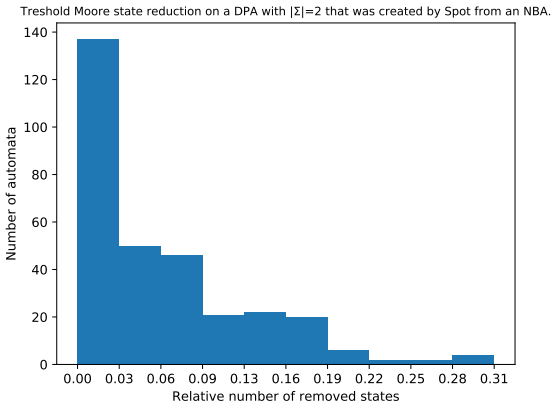
Like for the path refinement merger, we could use the internal equivalence relation of detnbaut to eliminate the need of explicitly calculating $\sim$. However, as we already barely achieved any reduction with $\equiv_L$ and "worse" relations only decrease the amount of reduction, this step seems obsolete.

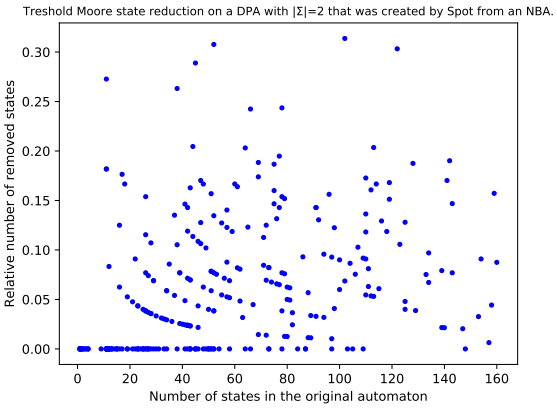
7.4 Different Thresholds

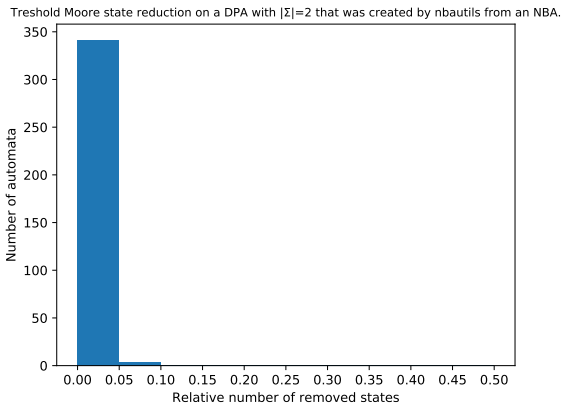
An interesting question to ask ourselves is whether changing $\equiv_M$ in the definition of $\equiv_{\mathrm{TM}}^{\sim}$ to a weaker relation still offers the same qualities, especially the language equivalence. One might especially look for a set of properties that a general equivalence relation $\sim'$ has to satisfy so that we can replace $\equiv_M$. As it turns out, this is not easy to do.











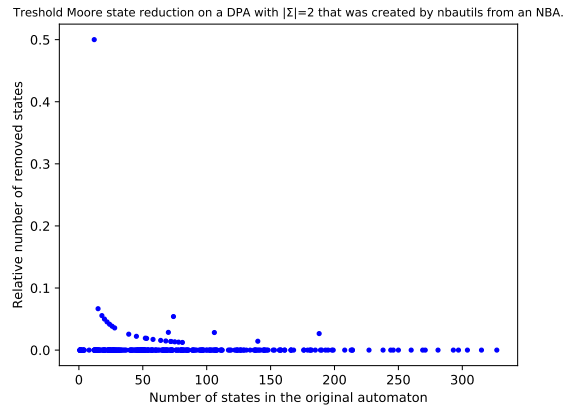


Figure 7.1: State reduction of different automata using $\mu_{\mathrm{TM}}^{\equiv_L}$.

Figure 7.2: Relative state reduction of different automata using $\mu_{\mathrm{TM}}^{\equiv_L}$.

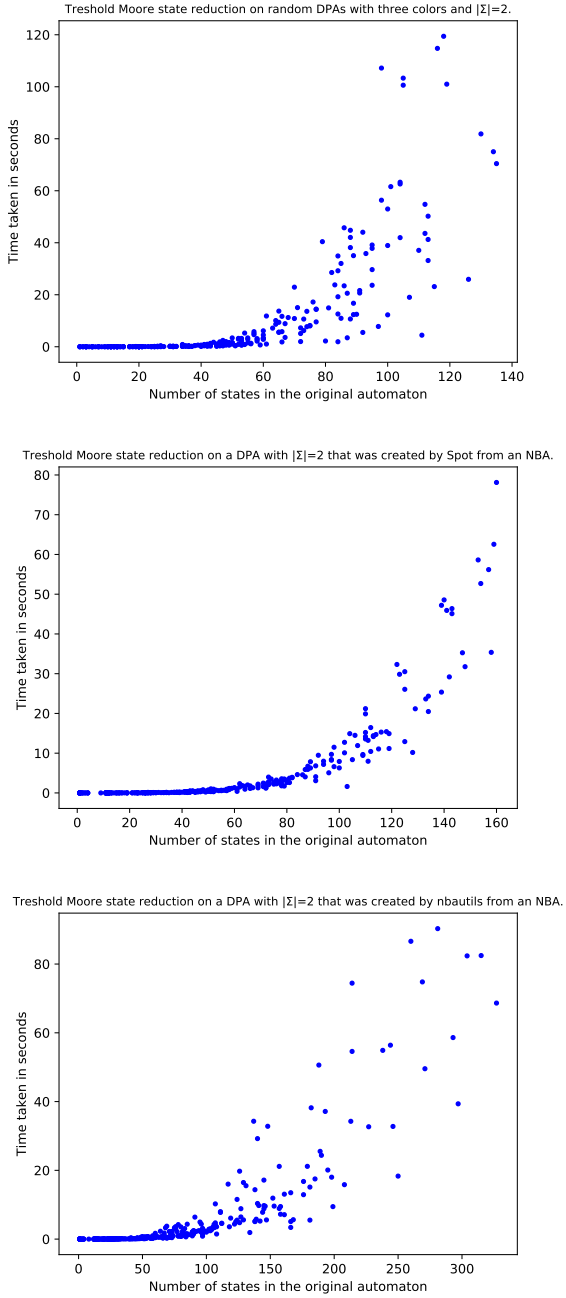


Figure 7.3: Time for state reduction of different automata using $\mu_{\mathrm{TM}}^{\equiv_L}$.

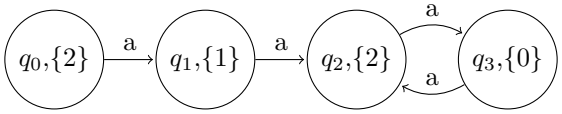


Figure 7.4: Example automaton which shows that using $\equiv_{\text{de}}$ for $\equiv_{\text{TM}}$ will not work.

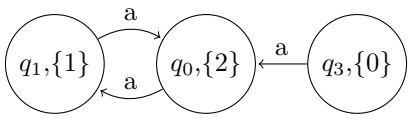


Figure 7.5: Example automaton which shows that using $\equiv_{\text{de}}$ for $\equiv_{\text{TM}}$ will not work.

In fact, figure 7.4 displays an automaton that shows that replacing $\equiv_M$ by $\equiv_{\text{de}}$ will not preserve language when used in a merge. For this purpose we will call this variation of $\equiv_{\text{Tde}}$ instead and fix $\sim = \equiv_L$ to be the complete language equivalence.

The equivalence classes are $\mathfrak{C}(\equiv_{\text{Tde}}) = \{\{q_0, q_2\}, \{q_1\}, \{q_3\}\}$, which means the automaton shown in figure 7.5 is a representative merge. However, this automaton rejects every word which is not the case for the original.

Chapter 8

Labeled SCC Filter

8.1 Theory

Definition 8.1.1. Let $\mathcal{A} = (Q_1, \Sigma, \delta_1, c_1)$ and $\mathcal{B} = (Q_2, \Sigma, \delta_2, c_2)$ be DPAs, $k \in \mathbb{N}$, and $\sim$ be an equivalence relation that implies language equivalence. We define a relation $\equiv_{\mathrm{LSF}}^{k,\sim}$ such that $(\mathcal{A}, p) \equiv_{\mathrm{LSF}}^{k,\sim} (\mathcal{B}, q)$ holds if and only if $(\mathcal{A}, p) \equiv_M^{\leq k} (\mathcal{B}, q)$ and $(\mathcal{A}, p) \sim (\mathcal{B}, q)$.

Lemma 8.1.1. *$\equiv_{LSF}^{k,\sim}$ is an equivalence relation.*

Proof. $\equiv_{\mathrm{LSF}}^{k,\sim}$ is the intersection of two equivalence relations and therefore one itself. □

Lemma 8.1.2. *If $l \leq k$ and $\sim \subseteq \approx$, then $\equiv_{LSF}^{k,\sim} \subseteq \equiv_{LSF}^{l,\approx}$.*

Proof. By Lemma 7.1.5, we have $\equiv_M^{\leq k} \subseteq \equiv_M^{\leq l}$. Thus, $\equiv_{\mathrm{LSF}}^{k,\sim}$ is an intersection of two sets which are respective subsets of the sets that intersect $\equiv_M^{\leq l}$. □

Definition 8.1.2. Let $\mathcal{A} = (Q, \Sigma, \delta, c)$ be a DPA. We define $\mathcal{A} \restriction_{>k}^{c} := \mathcal{A} \restriction_P$ with $P = \{q \in Q \mid c(q) > k\}$ and set $\preceq_k$ to be a total extension of the reachability preorder in $\mathcal{A} \restriction_{>k}^{c}$.

Definition 8.1.3. Let $\kappa \in \mathfrak{C}(\equiv_{\mathrm{LSF}}^{k,\sim}, \mathcal{A})$ be an equivalence class. We define $C_\kappa^k = \{r \in \kappa \mid c(r) > k \text{ and } r \text{ is } \preceq_k\text{-maximal among } \kappa\}$ and $M_\kappa^k = \kappa \setminus C_\kappa$.

Lemma 8.1.3. *Let $\mathcal{A}$ be a DPA and $\kappa \in \mathfrak{C}(\equiv_{LSF}^{k,\sim})$. Let $\mathcal{A}'$ be a representative merge of $\mathcal{A}$ w.r.t. M_κ by candidates C_κ. Then $(\mathcal{A}, p) \equiv_L (\mathcal{A}', p)$ for all p.*

Proof. Let r be the representative that is used in the merge. Let ρ and ρ' be the runs of $\mathcal{A}$ and $\mathcal{A}'$ on some α starting in p. We claim that ρ is accepting iff ρ' is accepting.

By Lemma 2.2.3, we know that $(\mathcal{A}, \rho(i)) \sim (\mathcal{A}, \rho'(i))$ and $(\mathcal{A}, \rho(i)) \equiv_M^{\leq k} (\mathcal{A}, \rho'(i))$ 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 $(\mathcal{A}, \rho(n)) \sim (\mathcal{A}, \rho'(n))$ and therefore $(\mathcal{A}, \rho(n)) \equiv_L (\mathcal{A}, \rho'(n))$.

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 ($(\mathcal{A}, \rho(i)) \equiv_M^{\leq k} (\mathcal{A}, \rho'(i))$) 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 $\mathcal{A} \restriction_{>k}^{c}$. 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 $\delta'(\rho'(n_2), \alpha(n_2)) = \delta'(\rho'(n_3), \alpha(n_3)) = r$, since all redirected transition target the representative state. We call $\delta(\rho'(n_3), \alpha(n_3)) = q$. Since between n_2 and n_3 no redirected transition is taken, $\rho'[n_2 + 1, n_3 + 1]$ is a valid path in $\mathcal{A}$, so we have $r \preceq_k q$ by choice of n_1. The fact that transitions to q are redirected to r however requires that $q \in M_\kappa$ and therefore q being not $\preceq_k$-maximal. Thus, there is a q' with $q \prec_k q'$ and therefore $r \prec_k q'$ which contradicts the choice of r from the set of candidates. □

Lemma 8.1.4. *Let $\mathcal{A}$ be a DPA and $\kappa \in \mathfrak{C}(\equiv_{LSF}^{k,\sim})$. Let $\mathcal{A}'$ be a representative merge of $\mathcal{A}$ w.r.t. M_κ by candidates C_κ. Then $(\mathcal{A}, p) \equiv_M^{\leq k} (\mathcal{A}', p)$ for all p.*

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 $c(\rho(i)) =^{\leq k} c'(\rho'(i))$ for all i which then proves the Lemma.

By Lemma 2.2.3, $(\mathcal{A}, \rho(i)) \equiv_M^{\leq k} (\mathcal{A}, \rho'(i))$ 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 $c(\rho(i)) =^{\leq k} c'(\rho'(i))$. □

While these are already good results, to go further requires us to restrict $\sim$ to $\equiv_L$.

Lemma 8.1.5. *Let $\mathcal{A}$ be a DPA and $\kappa \in \mathfrak{C}(\equiv_{LSF}^{k,\sim})$. Let $\mathcal{A}'$ be a representative merge of $\mathcal{A}$ w.r.t. M_κ by candidates C_κ. Then $(\mathcal{A}, p) \equiv_{LSF}^{k,\equiv_L} (\mathcal{A}', p)$ for all p.*

Proof. Representative merges never change priorities assigned to states. Together with Lemma 8.1.3 and Lemma 8.1.4 this already finishes the proof. □

We can now define a merger function using the LSF method and prove its correctness.

Definition 8.1.4. We define the *LSF merger function*:

$$\mu_{\mathrm{LSF}}^{k,\sim} : \{M_\kappa^k \mid \kappa \in \mathfrak{C}(\equiv_{\mathrm{LSF}}^{k,\sim}, \mathcal{A})\} \to 2^Q, M_\kappa^k \mapsto C_\kappa^k$$

Theorem 8.1.6. *Let $\mathcal{A}'$ be a representative merge of a DPA $\mathcal{A}$ w.r.t. $\equiv_{LSF}^{k,\equiv_L}$. Then $(\mathcal{A}, p) \equiv_{LSF}^{k,\equiv_L} (\mathcal{A}', p)$ for all p.*

Proof. Building a representative merge of $\mathcal{A}$ w.r.t. $\mu_{\mathrm{LSF}}^{k,\equiv_L}$ implicitly builds multiple representative merges $\mathcal{A}_0, \ldots, \mathcal{A}_m$, where $\mathcal{A}_{i+1}$ is a representative merge of $\mathcal{A}_i$ w.r.t. some M_κ^k by candidates C_κ^k. By Lemma 8.1.5, for all $j > i$, the states in κ_j are still pairwise $\mu_{\mathrm{LSF}}^{k,\equiv_L}$ equivalent. Moreover, κ_{i+1} is an $\mu_{\mathrm{LSF}}^{k,\equiv_L}$ equivalence class in $\mathcal{A}_i$. 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, by Lemma 8.1.5, satisfies $(\mathcal{A}, p) \equiv_{\mathrm{LSF}}^{k,\equiv_L} (\mathcal{A}', p)$. □

Corollary 8.1.7. *Every representative merge of a DPA $\mathcal{A}$ w.r.t. $\mu_{LSF}^{k,\equiv_L}$ is language equivalent to $\mathcal{A}$.*

A final result to show how versatile the LSF merge is is captured by the following Theorem.

Theorem 8.1.8. $\mu_{LSF}^{-1,\sim} = \mu_{skip}^{\sim}$

Proof. First, note that $\equiv_{\mathrm{LSF}}^{-1,\sim}$ is equivalent to $\sim$, as $\equiv_M^{\leq -1}$ is true for all pairs. Since $\mathcal{A} \restriction_{>-1}^{c}$ is the same as $\mathcal{A}$, the definitions align. □

8.2 Computation

Lemma 8.2.1. *For a given DPA $\mathcal{A}$ and equivalence relation $\sim$, $\mu_{LSF}^{k,\sim}$ can be computed in $\mathcal{O}(|Q| \cdot \log |Q|)$.*

Proof. By Lemma 7.2.1 we can compute $\equiv_M^{\leq k}$ in $\mathcal{O}(|Q| \cdot \log |Q|)$. Merging that relation with $\sim$ is possible in linear time, given a suitable data structure, and gives us $\equiv_{\mathrm{LSF}}^{k,\sim}$.

To construct $\mu_{\mathrm{LSF}}^{k,\sim}$ from $\equiv_{\mathrm{LSF}}^{k,\sim}$, the only non-trivial computation is that of C_κ^k. Building $\preceq_k$ is possible in linear time by Lemma 2.3.2. Given that ordering, the computation of C_κ^k and M_κ^k is a constant time operation for every κ. □

8.3 Efficiency

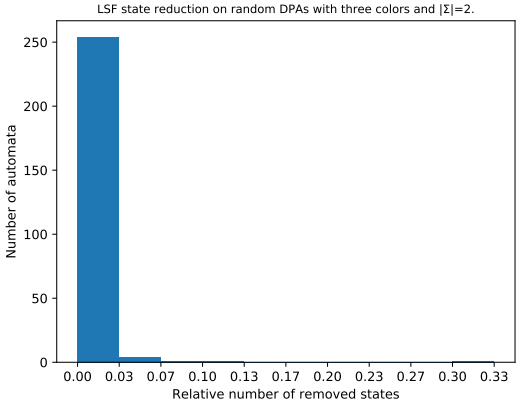
Figures 8.1 and 8.2 show the relative reduction achieved by the LSF merger. We iteratively applied $\mu_{\mathrm{LSF}}^{k,\equiv_L}$ with ascending k. For the detspot data set, a decent reduction of frequently above 10%, sometimes up to 55%, was achieved. For detnbaut, the best case was about 21%, not considering outliers. As before, the explicit computation of $\equiv_L$ dominates the run time (figure 8.3).

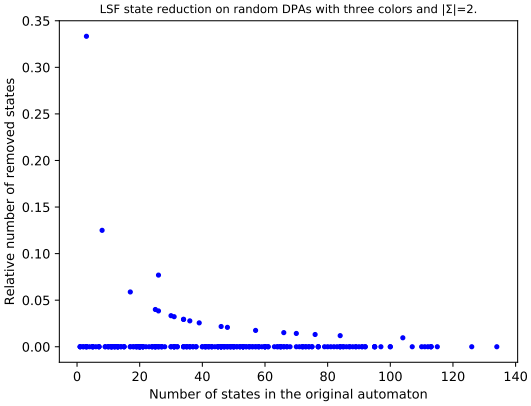
Again, we are able to use the equivalence relation given by nbautils instead of $\equiv_L$. This brings down the run time by a margin (figure 8.6). Unfortunately, the reduction gain already was in the lower numbers with $\equiv_L$ and is reduced further by weakening the relation. Figures 8.4 and 8.5 show that a reduction of more than 2% are the exception rather than the rule.

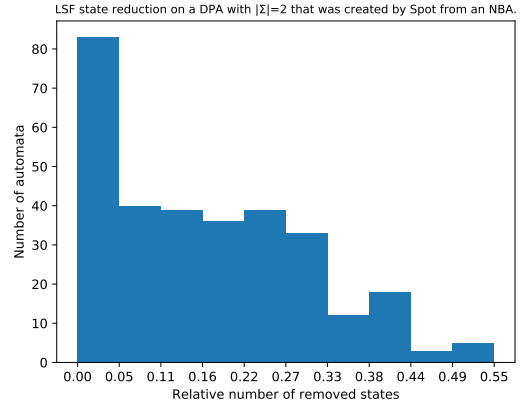
8.4 Different Thresholds

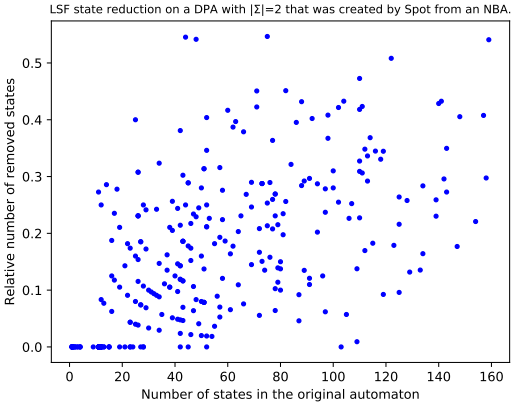
As we did for $\equiv_{\mathrm{TM}}^{\sim}$, the question whether we can use a more relaxed relation instead of $\equiv_M^{\leq k}$ arises. For example, we could define $\equiv_{\mathrm{deLSF}}^{k,\sim}$ to use $\equiv_{\mathrm{de}}^{\leq k}$ instead and $\mu_{\mathrm{deLSF}}^{k,\sim}$ analogously. As it turns out, this variant has similar issues as the same approach for $\equiv_{\mathrm{TM}}^{\sim}$.

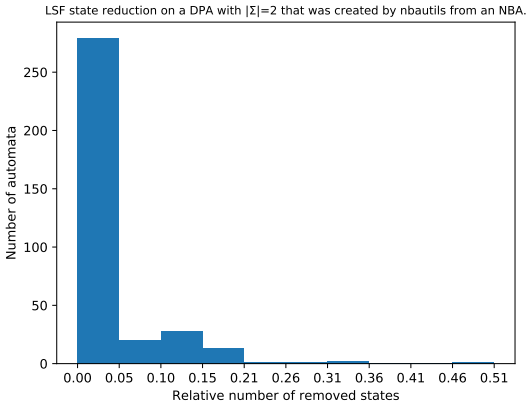
We can use the same example as before, which is again displayed in figure 8.7. The equivalence classes of $\equiv_{\mathrm{deLSF}}^{1,\equiv_L}$ are $\{\{q_0, q_2\}, \{q_1\}, \{q_3\}\}$. We can therefore merge $\kappa = \{q_0, q_2\}$ according to the defined rules. In $\mathcal{A} \restriction_{>1}^{c}$, q_0 and q_2 are not connected so we can remove either state and merge it into the other. That means figure 8.8 is a representative merge. However, this automaton rejects every word which is not the case for the original.











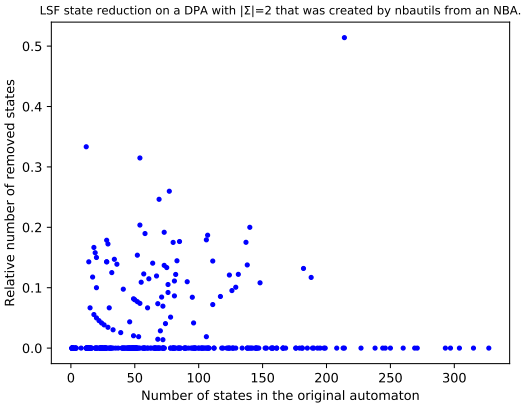


Figure 8.1: State reduction of different automata using $\mu_{\mathrm{LSF}}^{k,\equiv_L}$.

Figure 8.2: Relative state reduction of different automata using $\mu_{\mathrm{LSF}}^{k,\equiv_L}$.

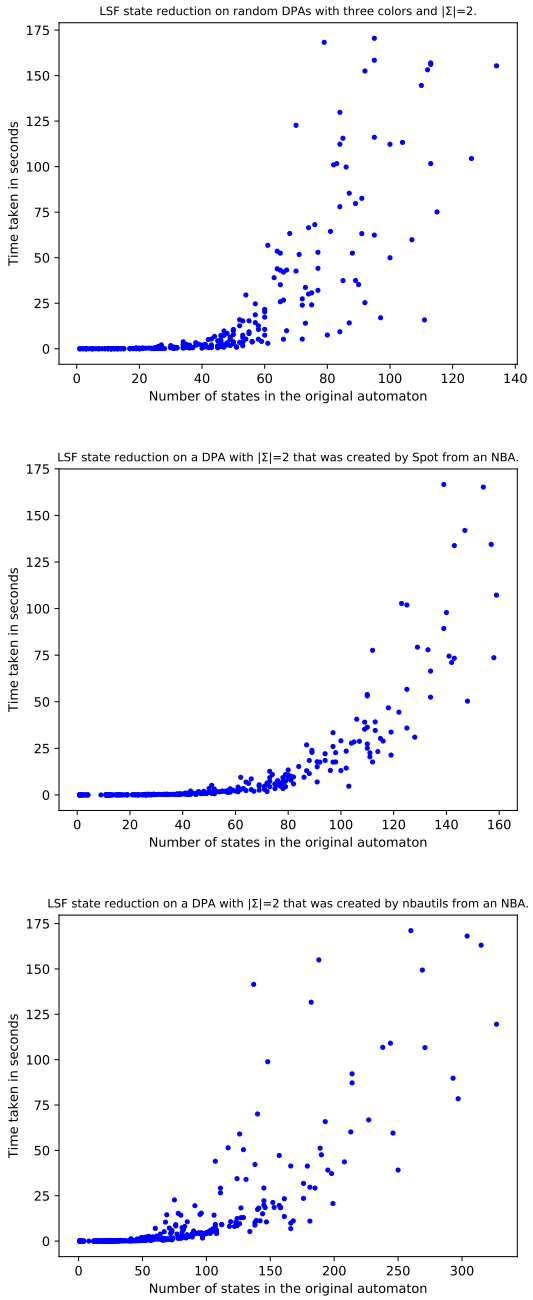


Figure 8.3: Time for state reduction of different automata using $\mu_{\mathrm{LSF}}^{k,\equiv_L}$.

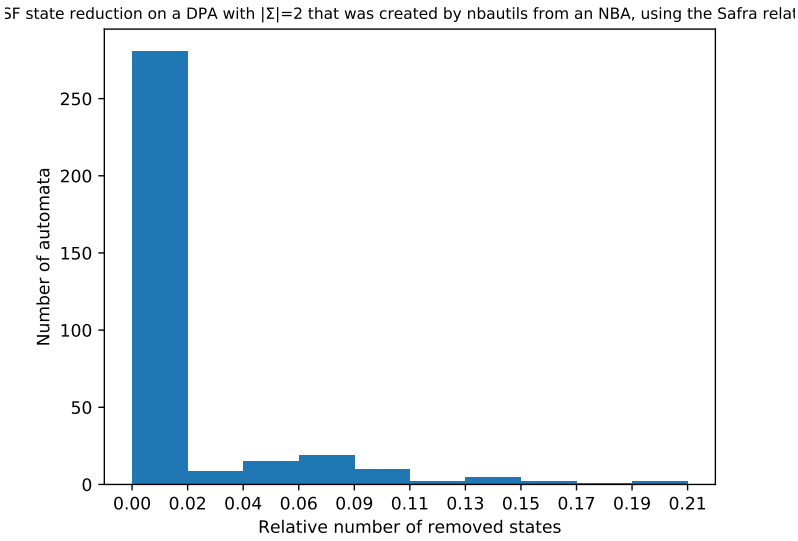


Figure 8.4: State reduction of different automata using $\mu_{\mathrm{LSF}}^{k,\sim}$.

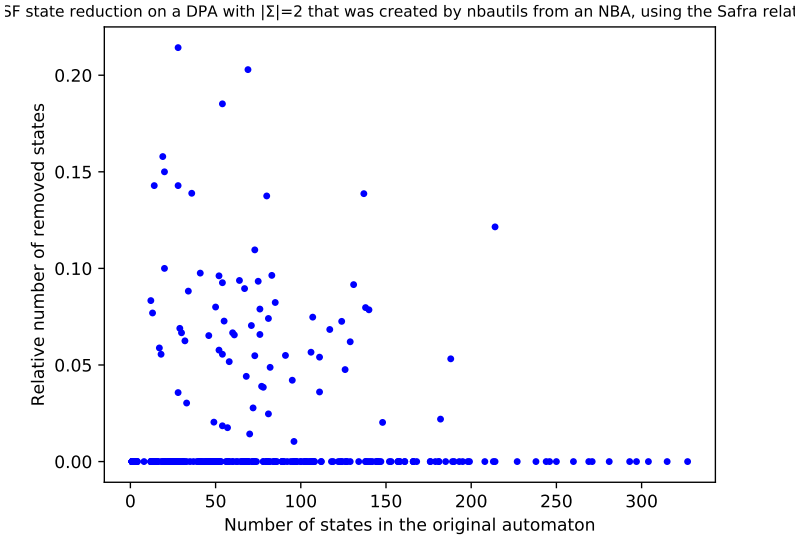


Figure 8.5: State reduction of different automata using $\mu_{\mathrm{LSF}}^{k,\sim}$.

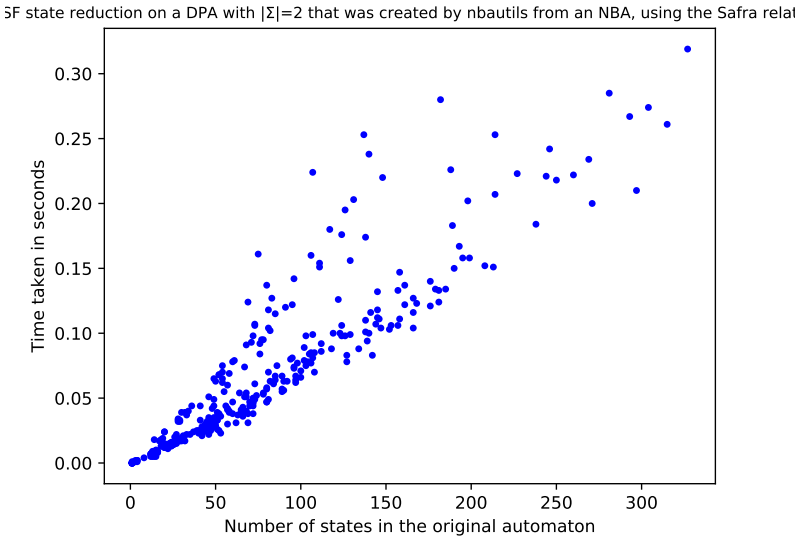


Figure 8.6: State reduction of different automata using $\mu_{\mathrm{LSF}}^{k,\sim}$.

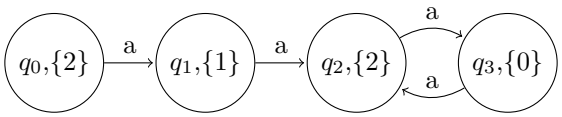


Figure 8.7: Example automaton which shows that using $\equiv_{\mathrm{de}}$ for $\equiv_{\mathrm{LSF}}$ will not work.

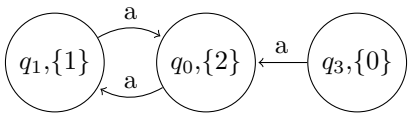


Figure 8.8: Example automaton which shows that using $\equiv_{\mathrm{de}}$ for $\equiv_{\mathrm{LSF}}$ will not work.

Chapter 9

Schewe

9.1 Theory

This section is based heavily on [20] and partially adapts their results from Büchi to parity automata.

Definition 9.1.1. Let $\mathcal{A} = (Q, \Sigma, \delta, c)$ be a DPA and let $\emptyset \neq C \subseteq M \subseteq Q$. Let $\mathcal{A}' = (Q', \Sigma, \delta', c')$ be another DPA. We call $\mathcal{A}'$ a *Schewe merge of $\mathcal{A}$ w.r.t. M by candidates C* if it satisfies the following:

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

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

Using the Schewe merge instead of the representative merge does not actually remove any additional states from the automaton, it only provides a better "framework" for following algorithms such as the Moore reduction. Example figure 9.1 shows that using a Schewe merge followed by another merge of μ_M creates a better end result.

The automaton has six states. Regarding Moore equivalence, every state builds a singleton class. For priority almost equivalence, the classes are $\{q_0, q_1\}$, $\{q_2, q_4\}$, and $\{q_3\}$. As all states within one equivalence class lie in the same SCC, a representative merge w.r.t. $\mu_{\text{skip}}^{\equiv_\dagger}$ will not change the number of states. However, a Schewe merge w.r.t. $\mu_{\text{skip}}^{\equiv_\dagger}$ can result in figure 9.2. Now, q_0 and q_1 are actually Moore equivalent and could be merged with μ_M.

Definition 9.1.2. Let $\mathcal{A}$ be DPA with a merger function $\mu : D \to 2^Q$. For a representative merge $\mathcal{A}'$, we define the *candidate relation* $\sim_{\mathcal{C}}^{\mu}$ as $p \sim_{\mathcal{C}}^{\mu} q$ iff there is a $C \in \mu(D)$ with $p, q \in C$.

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

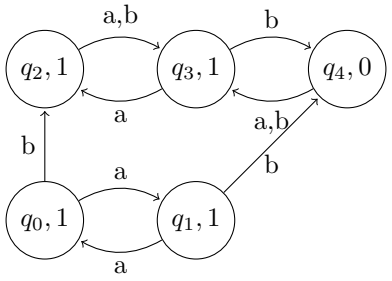


Figure 9.1: Example to show the effect of a Schewe merge.

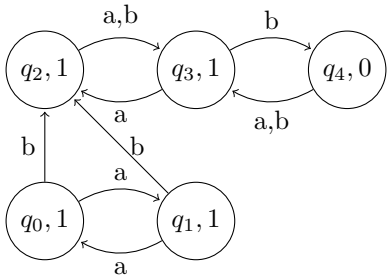


Figure 9.2: Automaton from figure 9.1 after Schewe merge.

Lemma 9.1.1. *Let $\mathcal{A}$ be a DPA and $\mu : D \to 2^Q$ be a merger function that is Schewe suitable. Let $\mathcal{A}'$ be a representative merge of $\mathcal{A}$ w.r.t. μ and let $\mathcal{A}''$ be the Schewe merge that uses the same choice of representatives. For all $p \sim_{\mathcal{C}}^{\mu} q$, $\delta'(p, a) \sim_{\mathcal{C}}^{\mu} \delta''(q, a)$.*

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

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

Lemma 9.1.2. *Let $\mathcal{A}$ be a DPA and $\mu : D \to 2^Q$ be a merger function that is Schewe suitable. Let $\mathcal{A}'$ be a representative merge of $\mathcal{A}$ w.r.t. μ and let $\mathcal{A}''$ be the Schewe merge that uses the same choice of representatives. Every run of $\mathcal{A}''$ has a suffix that is a run of $\mathcal{A}'$.*

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

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

Consider k_{l+1}. By definition of the Schewe merge, $\delta'(\rho(k_{l+1}), \alpha(k_{l+1})) \not\preceq_{\text{reach}}^{\mathcal{A}} \rho(k_{l+1})$. By Lemma 9.1.1, we know that $(\delta')^*(\rho(k_{l+1}), \alpha[k_{l+1}, k_{l'} + 1]) \sim_{\mathcal{C}}^{\mu} (\delta'')^*(\rho(k_{l+1}), \alpha[k_{l+1}, k_{l'} + 1]) =$

$\rho(k_{l'}+1)$. That means there is a state $r \in \kappa$ that is reachable from $\delta'(\rho(k_{l+1}), \alpha(k_{l+1}))$ in $\mathcal{A}$ and therefore from $\rho(k_l+1)$ as well.

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

Lemma 9.1.3. *Let $\mathcal{A}$ be a DPA and $\mu : D \to 2^Q$ be a merger function that is Schewe suitable. Let $\mathcal{A}'$ be a representative merge of $\mathcal{A}$ w.r.t. μ that was built with representatives $R \subseteq Q$. If a Schewe merge $\mathcal{A}''$ is built with the same representatives, then $(\mathcal{A}', p) \equiv_L (\mathcal{A}'', q)$ for all $p \sim_{\mathcal{C}}^{\mu} q$.*

Proof. Let $\alpha \in \Sigma^\omega$ be some word. Let ρ' be the run of $\mathcal{A}'$ on α starting in p and let ρ'' be the run of $\mathcal{A}''$ on α starting in q. Let $k_1, \ldots, k_n$ be the positions at which ρ'' uses a transition that is not present in $\mathcal{A}'$, i.e. $\rho''(k_i+1) \neq \delta'(\rho(k_i), \alpha(k_i))$. By Lemma 9.1.2, this list must be finite.

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

Assume that for some i, $\rho'(k_j+1) \sim_{\mathcal{C}}^{\mu} \rho''(k_j+1)$ is true for all $j < i$. As $\sim_{\mathcal{C}}^{\mu}$ is a congruence relation in $\mathcal{A}'$, $\rho'(k_i) \sim_{\mathcal{C}}^{\mu} \rho''(k_i)$ and $\rho'(k_i+1) = \delta'(\rho'(k_i), \alpha(k_i)) \sim_{\mathcal{C}}^{\mu} \delta'(\rho''(k_i), \alpha(k_i))$. By Lemma 9.1.1, $\delta'(\rho''(k_i), \alpha(k_i)) \sim \delta''(\rho''(k_i), \alpha(k_i)) = \rho''(k_i+1)$. □

9.2 Skip Merger

Lemma 9.2.1. *Let $\sim$ be a congruence relation that implies language equivalence. Then $\mu_{skip}^{\sim}$ is Schewe suitable.*

Proof. Let $\mathcal{A}'$ be a representative merge of a DPA $\mathcal{A}$ w.r.t. $\mu_{\text{skip}}^{\sim}$. We prove that $p \sim q$ iff $p \sim_{\mathcal{C}}^{\mu_{\text{skip}}^{\sim}} q$. The required properties then follow from the assumptions in the statement and from Lemma 3.1.2.

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

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

Corollary 9.2.2. *Let $\mathcal{A}$ be a DPA and let $\sim$ be a congruence relation that implies language equivalence. For each Schewe merge $\mathcal{A}'$ of $\mathcal{A}$ w.r.t. $\mu_{skip}^{\sim}$, $\mathcal{A} \equiv_L \mathcal{A}'$.*

Proof. Follows from Lemma 9.1.3, Lemma 9.2.1, and Theorem 3.1.4. □

For our final proof in this section we want to adapt a result from [20] to show a special relation between Schewe merges and priority almost equivalence.

Lemma 9.2.3. *Let $\mathcal{A}$ be a DPA, $\mathcal{S}$ be a Schewe merge of $\mathcal{A}$ w.r.t. $\mu_{skip}^{\equiv_\dagger}$, and $\mathcal{S}'$ be a representative merge of $\mathcal{S}$ w.r.t. μ_M. There is no smaller DPA than $\mathcal{S}'$ that is priority almost equivalent to $\mathcal{A}$.*

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

At first observe that $\mathcal{A} \equiv_\dagger \mathcal{S}'$: $\mathcal{A}$ and $\mathcal{S}$ are priority almost equivalent as for every state in $\mathcal{A}$, there is an equivalent representative in $\mathcal{S}$. $\mathcal{S}$ and $\mathcal{S}'$ are Moore equivalent by Lemma 2.2.4 and thus they are also priority almost equivalent by Theorem 2.1.13.

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

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

There is an SCC C in $\mathcal{S}'$ that contains all states in κ (argumentation is done similar to Lemma 3.1.2). Without loss of generality we can assume that likewise there is an SCC D in $\mathcal{B}$ that contains $f(\kappa)$. If no such SCC would exist, we could simply apply the Schewe merger to $\mathcal{B}$ to find an automaton that is smaller than $\mathcal{S}'$ and does have this property.

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

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

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

For each word $w \in \Sigma^*$, $c_{\mathcal{S}'}(\delta^*_{\mathcal{S}'}(p_1, w)) = c_\mathcal{B}(\delta^*_\mathcal{B}(q, w))$ or $\mathcal{S}'$ leaves C while reading w from p_1. The same holds for p_2. As p_1 and p_2 are distinct states in $\mathcal{S}'$, they cannot be Moore equivalent. That means there has to be a word u with $c_{\mathcal{S}'}(\delta^*_{\mathcal{S}'}(p_1, u)) \neq c_{\mathcal{S}'}(\delta^*_{\mathcal{S}'}(p_2, u))$. However, $\mathcal{S}'$ cannot leave C while reading u from p_1; if it did, it would reach a class λ and the transition would lead to the representative r_λ. As $\equiv_\dagger$ is a congruence relation, reading u from p_2 would lead to r_λ at the same position, so from that point on the two runs would be exactly the same.

Hence, $p'_1 = \delta^*_{\mathcal{S}'}(p_1, u)$ and $p'_2 = \delta^*_{\mathcal{S}'}(p_2, u)$ are still in C and $c_{\mathcal{S}'}(p'_1) \neq c_{\mathcal{S}'}(p'_2)$. That means at least one of these values must be different to $c_\mathcal{B}(\delta^*_\mathcal{B}(q, u))$, which contradicts our assumption.

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

We make sure that after reading any α_n from p, $\mathcal{S}'$ moves back to p. Let $\alpha_0 := \varepsilon$ and assume for induction that α_n has already been defined. After reading α_0, $\mathcal{S}'$ reaches p and $\mathcal{B}$ reaches some q. If $q \notin f(\kappa)$, i.e. $(\mathcal{S}', p) \not\equiv_\dagger (\mathcal{B}, q)$, there is a witness β and we can simply set $\alpha := \alpha_n\beta$. Otherwise, $q \in f(\kappa)$. With the claim we have proven, we can find a word w_q such that reading the word from p and q leads to states p' and q' that have different priorities but p' is still in C. Thus, there is a word u that leads back from p' to p. We then set $\alpha_{n+1} := \alpha_n w_q u$. This finishes our proof. □

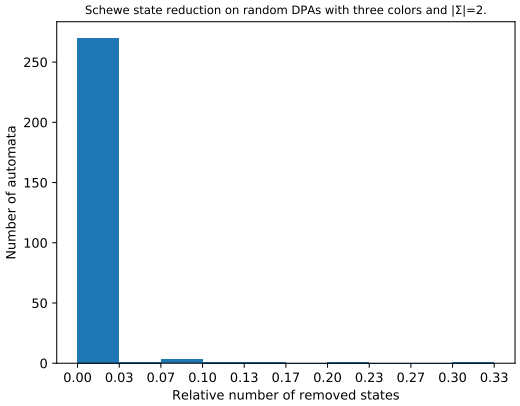
9.3 Computation

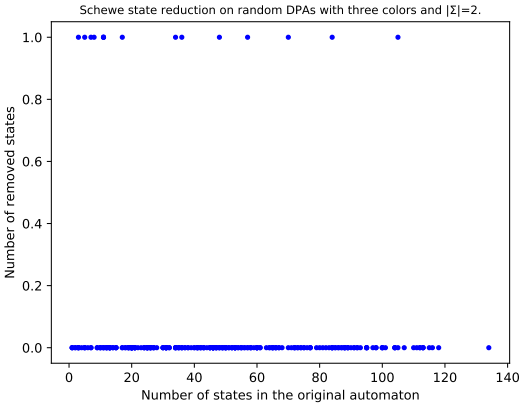
Lemma 9.3.1. *Given a DPA $\mathcal{A}$ and a merger function $\mu : D \to 2^Q$ in suitable data structures, a Schewe merge of $\mathcal{A}$ w.r.t. μ can be computed in linear time.*

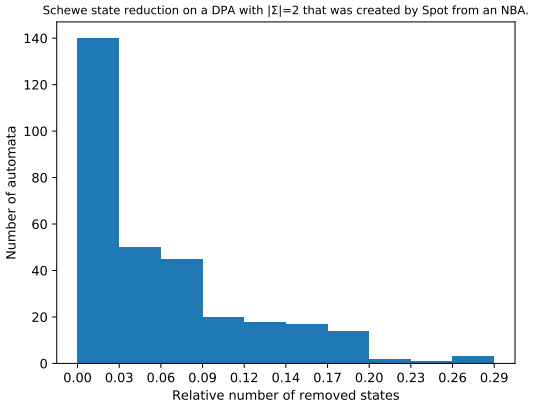
Proof. In addition to the argumentation provided in Lemma 2.2.2, a Schewe merge has the additional question of "is p reachable from q". This can be answered in linear time by computing the SCCs of the automaton once and using a lookup table to provide constant time queries. □

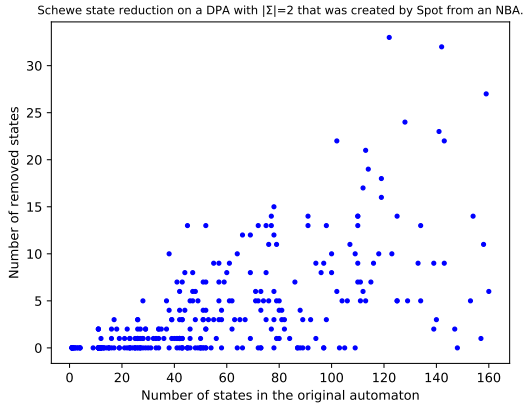
9.4 Efficiency

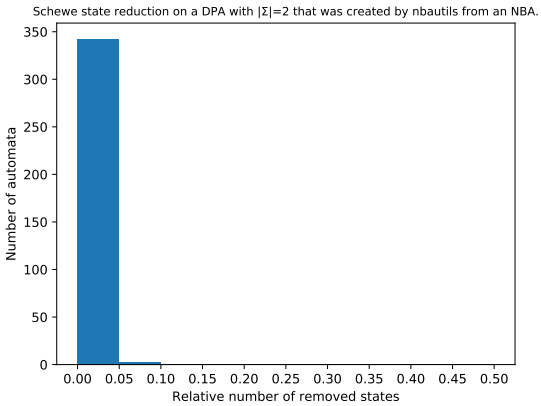
Figures 9.3 and 9.4 show the effect of building Schewe merges w.r.t. the skip merger. After the Schewe merge was constructed, we also built a representative merge w.r.t. μ_M to see the difference of the Schewe merge compared to a normal representative merge. Unfortunately, the reduction is barely any better than that achieved by building a merge w.r.t. μ_M directly (figure 2.3). The reasoning here is the same as that which was given in chapter 3 already to explain the bad results of $\mu^{\sim}_{\text{skip}}$: our testing automata contain only few SCCs in general which limits the potential of a Schewe merge.











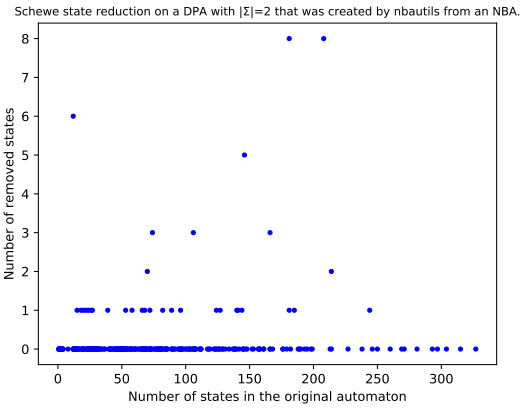


Figure 9.3: State reduction of different automata using $\mu_{\text{skip}}^{\equiv_\dagger}$.

Figure 9.4: Relative state reduction of different automata using $\mu_{\text{skip}}^{\equiv_\dagger}$.

Chapter 10

Combined Results

For the final chapter of the thesis, we want to provide a small insight into the effect of combining all reduction techniques that were presented. To be precise, we build several consecutive merges of an input DPA in the following order:

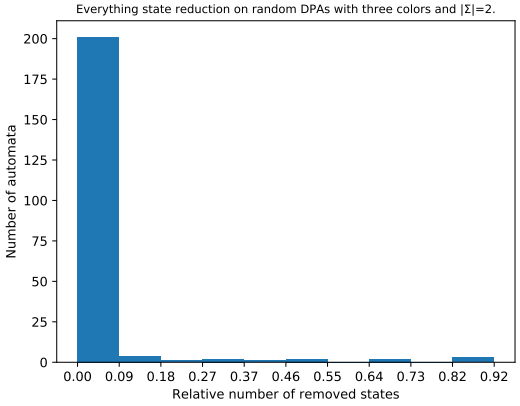
1. Normalize c.
2. Schewe merge w.r.t. $\mu_{\text{skip}}^{\equiv_L}$.
3. Representative merge w.r.t. μ_M.
4. Representative merge w.r.t. μ_{IM}.
5. Representative merge w.r.t. $\mu_{\text{TM}}^{\equiv_L}$.
6. Representative merge w.r.t. $\mu_{\text{LSF}}^{k,\equiv_L}$ for all k.
7. Representative merge w.r.t. μ_{de}.
8. Representative merge w.r.t. $\mu_{\text{PR}}^{\lambda}$ for all classes λ of $\equiv_L$.

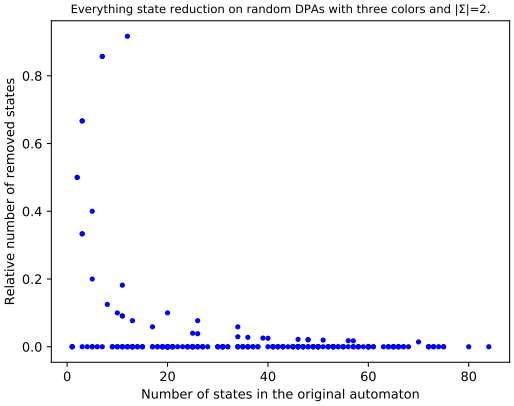
Figures 10.1 and 10.2 show the usual plots for efficiency and figure 10.3 shows the required run time. We were actually able to reuse $\equiv_L$ after its first computation to safe a large amount of time. It anyway quickly becomes unfeasible to perform a reduction like this for growing automata.

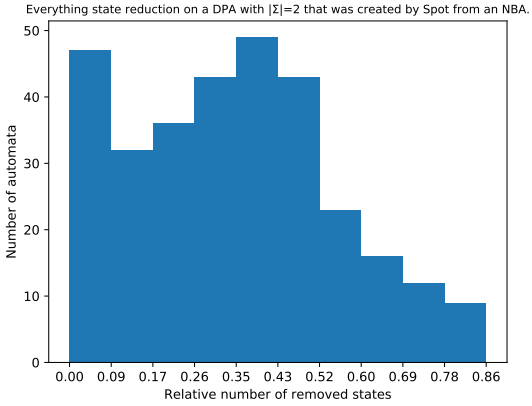
The number of reduced states shows a satisfying result on detspot and detnbaut, with a majority of the automata witnessing a reduction of 10% or more.

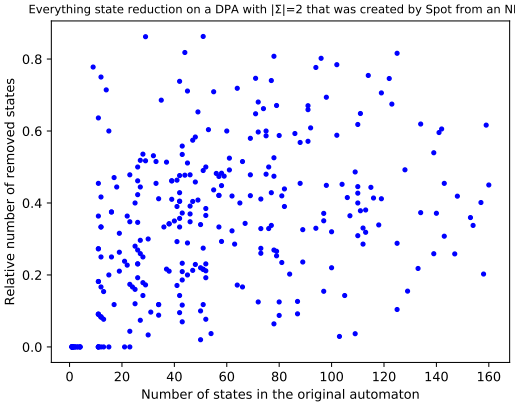
We can see the effect of changing the size of Σ in figures 10.4 and 10.5. As one might expect, the run time is higher for larger alphabets; we assume that there is a linear correlation but there is not enough data to confidently support that theory.

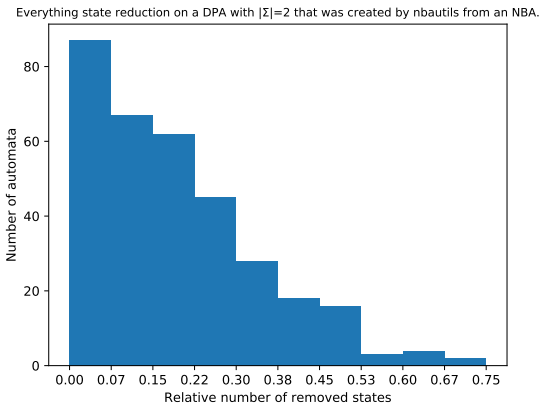
Similarly, doubling the size of the alphabet seems to halve the efficiency of the reduction. Again, the data is not conclusive enough to support that argument with absolute certainty.











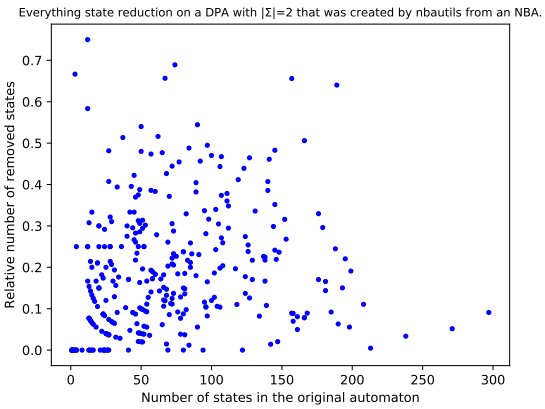


Figure 10.1: State reduction of different automata.

Figure 10.2: Relative state reduction of different automata.

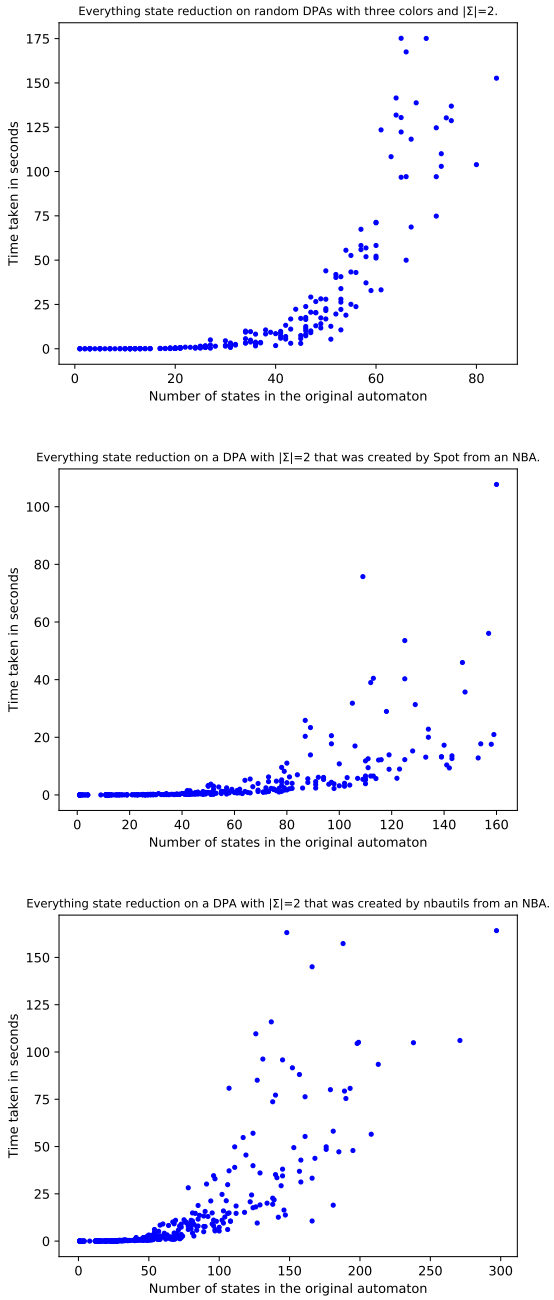


Figure 10.3: Time for state reduction of different automata.

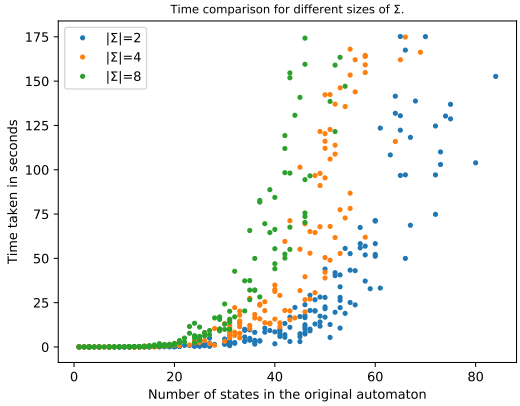


Figure 10.4: Compare run time for different sizes of alphabets.

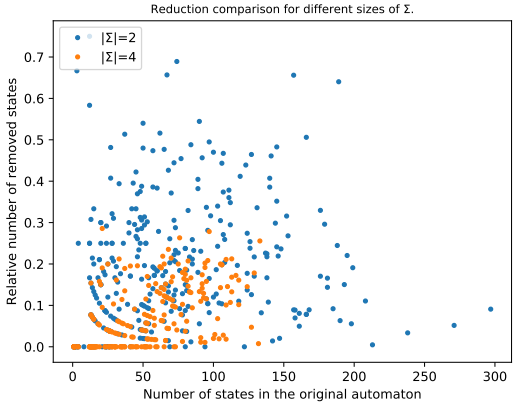


Figure 10.5: Compare relative reduction for different sizes of alphabets.

Appendices

Appendix A

Examples

A.1 Skip merger

Figure A.1 shows an automaton. If we use $\sim = \equiv_L$, then both states are $\sim$-equivalent. As q_0 is not reachable from q_1, the skip merger can remove q_0 to get to the automaton in figure A.2.

A.2 Delayed Simulation

Figure A.3 shows an automaton. The entire delayed simulation automaton is of size 75, so we only consider one pair of states for the example. To determine whether $q_0 \leq_{\text{de}} q_1$ is true, we need the part of $\mathcal{A}_{\text{de}}$ that is displayed in figure A.4. The question to check is whether from $(q_0, q_1, \checkmark)$, every word is accepted.

As we can see, the run induced by ba^ω ends in the loop $(q_3, q_3, 0)(q_2, q_2, 0)$ which is not accepting. Thus, $q_0 \not\leq_{\text{de}}^{ba^\omega} q_1$ and therefore $q_0 \not\leq_{\text{de}} q_1$. This coincides with the definition of delayed simulation: if the run starting in q_0 once sees priority 0, afterwards neither run will see that priority again, as they are stuck in the loop $q_3 q_2$.

A.3 Iterated Moore

Figure A.5 shows our example automaton for μ_{IM}. We have $q_0 \not\equiv_M q_1$, as the two states have different priorities. With iterated Moore equivalence, we actually have $q_0 \equiv_{IM} q_1$:

First, we choose our order of SCCs as $S_0 = \{q_0\}$ and $S_1 = \{q_1\}$. $\mathcal{B}_1$ then has only one state q_1 with the transition to itself. $\mathcal{B}'_0$ is then the same automaton as $\mathcal{A}$. S_0 is a trivial SCC, it is not M'_0-equivalent to any other state and $(\delta'_0(q_0, a), \delta'_0(q_1, a)) \in M'_0$ for all a. Therefore, $c_0(q_0)$ is set to 1 and we have $(q_0, q_1) \in M_0$. This results in figure A.6.

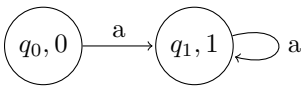


Figure A.1: Example for the skip merger.

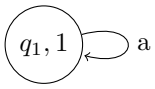


Figure A.2: Example for the skip merger.

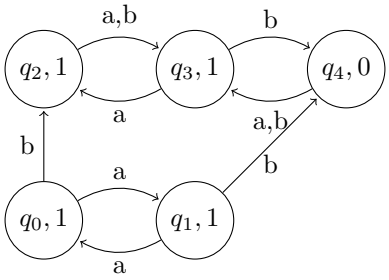


Figure A.3: Example for the delayed simulation merger.

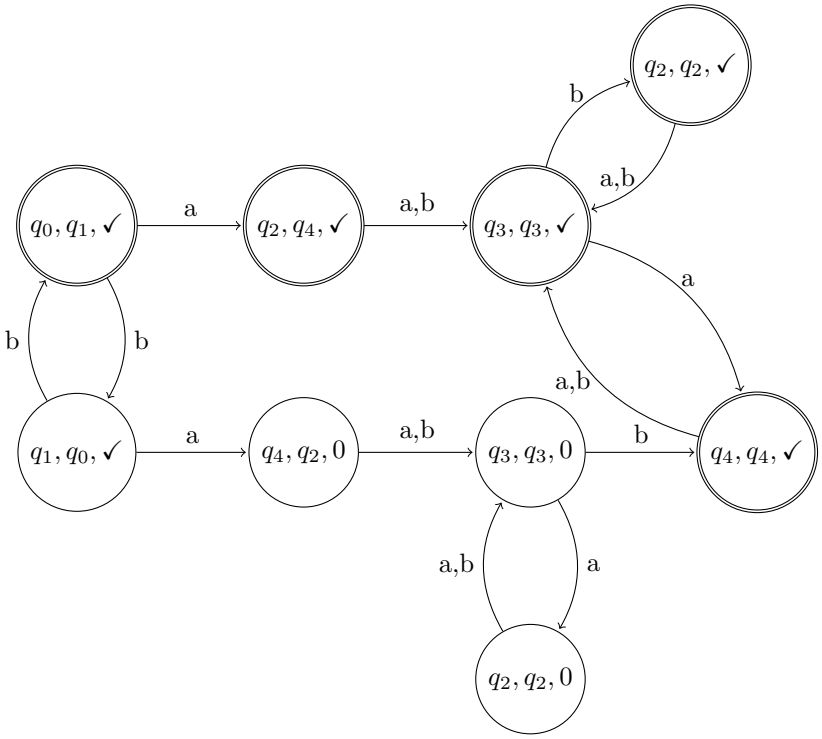


Figure A.4: Example for the delayed simulation merger. (Delayed simulation game)

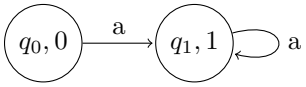


Figure A.5: Example for the iterated Moore merger.

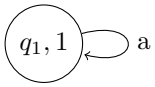


Figure A.6: Example for the iterated Moore merger.

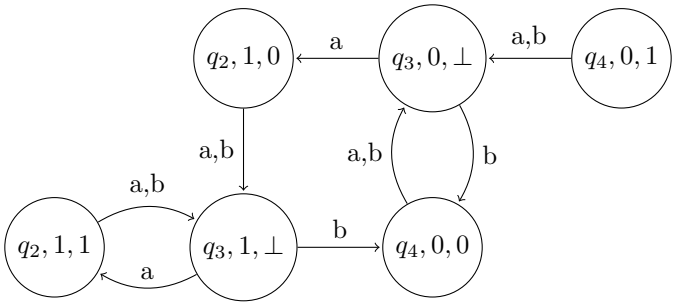


Figure A.7: Example for the path refinement merger.

A.4 Path Refinement

We use the same example automaton as for delayed simulation, A.3. $\equiv_L$ has the three equivalence classes $\{q_0, q_1\}$, $\{q_2, q_4\}$, and $\{q_3\}$. We choose to merge by $\lambda = \{q_2, q_4\}$.

Figure A.7 displays the transition structure $\mathcal{A}^{\lambda}_{\text{visit}}$. The relation V then consists of three equivalence classes: $\{(q_3, 1, \bot), (q_3, 0, \bot)\}$, $\{(q_2, 1, 0), (q_4, 0, 0)\}$, and $\{(q_2, 1, 1), (q_4, 0, 1)\}$. We are interested in the question whether $\iota^1_{q_2} = (q_2, 1, 1)$ and $\iota^1_{q_4} = (q_4, 0, 1)$ are equivalent.

Unfortunately, V_M, the congruence refinement of V, actually only contains singleton classes, so $q_2 \not\equiv^{\lambda}_{\text{PR}} q_4$. This is because $\delta^*_{\text{visit}}(\iota^1_{q_2}, aa) = (q_2, 1, 1)$ and $\delta^*_{\text{visit}}(\iota^1_{q_4}, aa) = (q_2, 1, 0)$, which is not a pair in V.

This shows that by reading aa from q_2, the automaton reaches back to λ and visits priorities of at least 1. On the other hand, from q_4 reading aa sees priority 0 once.

A.5 Threshold Moore

Figure A.8 shows an example DPA. All states are language equivalent; to be specific, they all accept the language $\Sigma^* b^\omega$. If we consider $\equiv^{\leq 0}_M$, we find that $q_0 \equiv^{\leq 0}_M q_1$ and therefore also $q_0 \equiv^{\equiv_L}_{\text{TM}} q_1$.

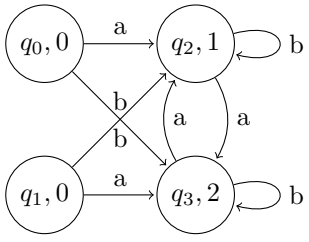


Figure A.8: Example for the Threshold Moore merger.

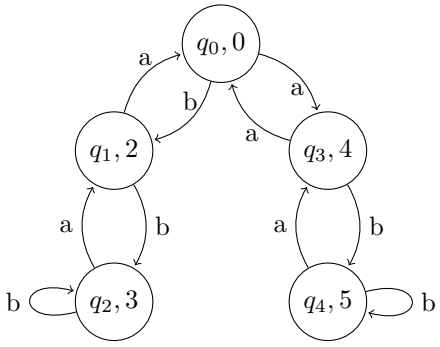


Figure A.9: Example for the LSF merger.

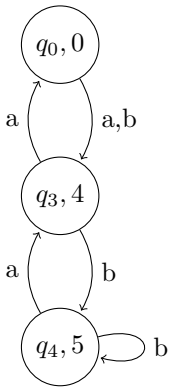


Figure A.10: Example for the LSF merger.

A.6 LSF

For the LSF merger, we use figure A.9 as an example. As parameters, we choose $\sim = \equiv_L$ and $k = 1$.

All states are language equivalent: from any state, the language $(\Sigma^* a)^\omega$ is accepted. Also, $q_1 \equiv_M^{\leq 1} q_3$ and $q_2 \equiv_M^{\leq 1} q_4$. Thus, the equivalence classes of $\equiv_{\mathrm{LSF}}^{1, \equiv_L}$ are $\{q_0\}$, $\{q_1, q_3\}$, and $\{q_2, q_4\}$.

One possible choice for $\preceq_k$ is $q_1 \simeq_k q_2 \prec_k q_3 \simeq_k q_4$. Thus, we can merge these states and obtain what is shown in figure A.10.

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] 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.

[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.

[8] 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.

[9] 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.

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

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

[12] 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.

[13] Max Michel. Complementation is much more difficult with automata on infinite words. 1988.

[14] 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.

[15] A.W. Mostowski. Hierarchies of weak automata and weak monadic formulas. *Theoretical Computer Science*, 83(2):323 – 335, 1991.

[16] Anton Pirogov. nbautils. `https://github.com/apirogov/nbautils`, 2018.

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

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

[19] Sven Schewe. Tighter bounds for the determinisation of büchi automata. In Luca de Alfaro, editor, *Foundations of Software Science and Computational Structures*, pages 167–181, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[20] 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.

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

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

[23] Wolfgang Thomas. Handbook of formal languages, vol. 3. chapter Languages, Automata, and Logic, pages 389–455. Springer-Verlag New York, Inc., New York, NY, USA, 1997.

[24] Andreas Tollkötter. Master thesis. `https://github.com/Wurstinator/master-thesis`, 2018.