Chapter 1

Basic Definitions

The first chapter defines fundamentals of this thesis and notation used later on.

1.1 General Mathematical Terms

As our main focus is ω-words, we will require a small extension of natural numbers into the transfinite realm.

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 powerset 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 $\mathfrak{C}(R) = \{[x]_R \mid x \in R\}$.

1.2 Words and Languages

Definition 1.2.1. 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 $\Sigma^* = \bigcup\limits_{n \in \mathbb{N}} \Sigma^n$ as the set of all finite words over Σ.

Definition 1.2.2. 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 1.2.3. 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 1.2.4. 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 1.2.5. 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 1.2.6. Let $w \in \Sigma^* \cup \Sigma^\omega$ be a word. We define a substring or subword 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 1.2.7. 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 1.2.8. The *occurrence set* of a word $w \in \Sigma^* \cup \Sigma^\omega$ is the set of symbols which occur at least once in w.

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

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

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

1.3 Automata

Definition 1.3.1. 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 1.3.2. 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 is the set of all words and ω-words that are accepted by $\mathcal{A}$.

Definition 1.3.3. 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, we write $L(\mathcal{A})$.

Definition 1.3.4. 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, we write $L(\mathcal{A})$.

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 1.3.5. 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 1.3.6. 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 1.3.7. Let $\mathcal{S} = (Q, \Sigma, \delta)$ be a deterministic transition structure and let $R \subseteq Q \times Q$ be an equivalence relation over Q. We call R a *congruence relation* if for all $(p, q) \in R$ and all $a \in \Sigma$, also $(\delta(p, a), \delta(q, a)) \in R$.

Chapter 2

Theory

2.1 General Results

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

2.1.1 Equivalence Relations

In general, we use the symbol $\equiv$ to denote equivalence relations, mostly between states of an automaton. 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
- Path refinement equivalence, $\equiv_{\mathrm{PR}}$. Defined in
- Threshold Moore equivalence, $\equiv_{\mathrm{TM}}$. Defined in
- Labeled SCC filter equivalence, $\equiv_{\mathrm{LSF}}$. Defined in

Language Equivalence

Definition 2.1.1. Let $\mathcal{A}$ be an ω-automaton with state set Q. We define *language equivalence* over Q as $p \equiv_L q$ if and only if for all words $\alpha \in \Sigma^\omega$, $\mathcal{A}$ accepts α from p iff $\mathcal{A}$ accepts α from q.

Lemma 2.1.1. *$\equiv_L$ is a congruence relation.*

Proof. It is obvious that $\equiv_L$ is an equivalence relation. For two states $p \equiv_L q$ and some successors $p' = \delta(p, a)$ and $q' = \delta(q, a)$, it must be true that $p' \equiv_L q'$. 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 $p \not\equiv_L q$. $\square$

Priority Almost Equivalence

Definition 2.1.2. Let $\mathcal{A} = (Q, \Sigma, \delta, c)$ be a DPA. We define *priority almost equivalence* over Q as $p \equiv_\dagger q$ if and only if for all words $\alpha \in \Sigma^\omega$, $c^*(p, \alpha)$ and $c^*(q, \alpha)$ differ at only finitely many positions.

Lemma 2.1.2. *Priority almost equivalence is a congruence relation.*

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

Lemma 2.1.3. *Priority almost equivalence of a given DPA can be computed in quadratic time.*

Proof. $\square$

Definition 2.1.3. Let $\mathcal{A} = (Q_1, \Sigma, q_0^1, \delta_1, c_1)$ and $\mathcal{B} = (Q_2, \Sigma, q_0^2, \delta_2, c_2)$ be DPAs. We define the deterministic Büchi automaton $\mathcal{A} \intercal \mathcal{B} = (Q_1 \times Q_2, \Sigma, (q_0^1, q_0^2), \delta_\intercal, F_\intercal)$ 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 2.1.4. *Let $\mathcal{A} = (Q_1, \Sigma, q_0^1, \delta_1, c_1)$ and $\mathcal{B} = (Q_2, \Sigma, q_0^2, \delta_2, c_2)$ be DPAs. $\mathcal{A}$ and $\mathcal{B}$ are priority almost-equivalent iff $L(\mathcal{A} \intercal \mathcal{B}) = \emptyset$.*

Proof. For the first direction of implication, let $L(\mathcal{A} \intercal \mathcal{B}) \neq \emptyset$, so there is a word α accepted by that automaton. Let $(p_0, q_0)(p_1, q_1)\cdots$ be the accepting run on α. Then $p_0 p_1 \cdots$ and $q_0 q_1 \cdots$ 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 $\mathcal{A}$ and $\mathcal{B}$ being not priority almost-equivalent.

For the second direction, let $\mathcal{A}$ and $\mathcal{B}$ 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. □

Lemma 2.1.5. *The priority almost-equivalence of a DPA $\mathcal{A}$ can be computed in $\mathcal{O}(|\mathcal{A}|^2)$.*

Proof. The definition of $\mathcal{A} \intercal \mathcal{A}$ is a straightforward construction and can be done in quadratic time. By Lemma 2.1.4, we know that p and q are priority almost-equivalent iff the language of $(\mathcal{A} \intercal \mathcal{A})_{(p,q)}$ is empty.

The product automaton $\mathcal{A} \intercal \mathcal{A}$ itself can be computed in quadratic time. The SCCs of $\mathcal{A} \intercal \mathcal{A}$ can be computed in linear time in said automaton, i.e. in $\mathcal{O}(|\mathcal{A}|^2)$.

The language from a given starting state is non-empty iff a goal SCC is reachable, i.e. an SCC that contains an accepting state and is non-trivial. It is a rather simple procedure to collect these SCCs and as described in the proof of Lemma 2.2.1, we can merge all of them into a single state $\hat{q}$ in $\mathcal{O}(|\mathcal{A}^2|)$.

All that remains to be done is to find all states from which $\hat{q}$ is unreachable. This is easily done by using the transposed automaton of $\mathcal{A} \intercal \mathcal{A}$, which has all transition edges inverted, and performing a DFS starting in $\hat{q}$. States which are **not** found here correspond to priority almost-equivalent pairs. □

Moore Equivalence

Definition 2.1.4. Let $\mathcal{A} = (Q, \Sigma, \delta, c)$ be a DPA. We define *Moore equivalence* over Q as $p \equiv_M q$ if and only if for all words $w \in \Sigma^*$, $c(\delta^*(p, w)) = c(\delta^*(q, w))$.

Lemma 2.1.6. *$\equiv_M$ is a congruence relation.*

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

Lemma 2.1.7. *Moore equivalence of a given DPA can be computed in log-linear time.*

Proof. □

Lemma 2.1.8. $\equiv_M \subseteq \equiv_\dagger \subseteq \equiv_L$

Proof. Let $\mathcal{A} = (Q_\mathcal{A}, \Sigma, q_0^\mathcal{A}, \delta_\mathcal{A}, c_\mathcal{A})$ and $\mathcal{B} = (Q_\mathcal{B}, \Sigma, q_0^\mathcal{B}, \delta_\mathcal{B}, c_\mathcal{B})$ be two DPA that are priority almost-equivalent and assume towards a contradiction that they are not language equivalent. Due to symmetry we can assume that there is a $w \in L(\mathcal{A}) \setminus L(\mathcal{B})$.

Consider $\alpha = \lambda_\mathcal{A}(q_0^\mathcal{A}, w)$ and $\beta = \lambda_\mathcal{B}(q_0^\mathcal{B}, w)$, the priority outputs of the automata on w. By choice of w, we know that $a := \max \mathrm{Inf}(\alpha)$ is even and $b := \max \mathrm{Inf}(\beta)$ is odd. Without loss of generality, assume $a > b$. That means a is seen only finitely often in β but infinitely often in a. Hence, α and β differ at infinitely many positions where a occurs in α. That would mean w is a witness that the two automata are not priority almost-equivalent, contradicting our assumption. $\square$

2.1.2 Representative Merge

Definition 2.1.5. Let $\mathcal{A} = (Q, \Sigma, \delta, c)$ be a DPA and let $\emptyset \neq C \subseteq M \subseteq Q$. Let $\mathcal{A}' = (Q', \Sigma, \delta', c')$ be another DPA. We call $\mathcal{A}'$ a *representative merge of $\mathcal{A}$ w.r.t. M by candidates C* if it satisfies the following:

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

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

Definition 2.1.6. Let $\mathcal{A} = (Q, \Sigma, \delta, c)$ be a DPA and let $\mu : D \to (2^Q \setminus \emptyset)$ be a function for some $D \subseteq 2^Q$. If all sets in D are pairwise disjoint and for all $X \in D$, $\mu(X) \subseteq X$, we call μ a *merger function*.

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

A common special case of this are quotient automata that are often used in state space reduction. Given a congruence relation $\sim$, the quotient automaton w.r.t. $\sim$ is equivalent to a representative merge w.r.t. $\mu : \mathfrak{C}(\sim) \to 2^Q, \kappa \mapsto \kappa$.

Lemma 2.1.9. *Let $\mathcal{A} = (Q, \Sigma, \delta, c)$ be a DPA and let $\sim$ be an equivalence relation. A representative merge of $\mathcal{A}$ w.r.t. $\sim$ is a representative merge of $\mathcal{A}$ w.r.t. $\mu : \mathfrak{C}(\sim) \to 2^Q, \kappa \mapsto \kappa$.*

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

Lemma 2.1.10. *Let $\mathcal{A} = (Q, \Sigma, \delta, c)$ be a DPA and let $M_1, M_2 \subseteq Q$. Let $\mathcal{A}_1$ be a representative merge of $\mathcal{A}$ w.r.t. M_1 by some candidates C_1. Let $\mathcal{A}_{12}$ be a representative merge of $\mathcal{A}_1$ w.r.t. M_2 by some candidates C_2. If M_1 and M_2 are disjoint, then there is a representative merge $\mathcal{A}_2$ of $\mathcal{A}$ w.r.t. M_2 by candidates C_2 such that $\mathcal{A}_{12}$ is a representative merge of $\mathcal{A}_2$ w.r.t M_1 by candidates C_1.*

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

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

Lemma 2.1.11. *Let $\mathcal{A}$ be a DPA. Let $\sim$ be a congruence relation on Q and let $M \subseteq Q$ such that for all $x, y \in M$, $x \sim y$. Let $\mathcal{A}'$ be a representative merge of $\mathcal{A}$ w.r.t. M by candidates C. Let ρ and ρ' be runs of $\mathcal{A}$ and $\mathcal{A}'$ on some α. Then for all i, $\rho(i) \equiv \rho'(i)$.*

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

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

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

- Moore merger μ_M. Defined below.
- Schewe merger, $\mu_{\text{Sch}}^{\sim}$. Defined in

Definition 2.1.7. Let $\mathcal{A}$ be a DPA. The Moore merger μ_M is defined as $\mu_M : \mathfrak{C}(\equiv_M) \to 2^Q, \kappa \mapsto \kappa$.

Lemma 2.1.12. *Let $\mathcal{A}$ be a DPA and let $\mathcal{A}'$ be a representative merge of $\mathcal{A}$ w.r.t. μ_M. Then $\mathcal{A}$ and $\mathcal{A}'$ are language equivalent.*

Proof. $\square$

2.1.3 Reachability

Definition 2.1.8. 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 only have one connected component, i.e. for all states p and q, there is a state r such that p and q are both reachable from r. In practice, most automata have an predefined initial state and a simple depth first search can be used to eliminate all unreachable states.

Lemma 2.1.13. *$\preceq_{\text{reach}}^{\mathcal{S}}$ is a preorder.*

Definition 2.1.9. 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 2.1.14. *For a given deterministic transition structure $\mathcal{S}$, a total extension of reachability is computable in $\mathcal{O}(|\mathcal{S}|)$.*

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

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

2.2 Schewe

This section is based heavily on [12] and mostly adapts their results from Büchi to parity automata.

Definition 2.2.1. 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 *Schewe merger function* $\mu^{\sim}_{\text{Sch}} : D \to 2^Q$ as follows: for each equivalence class $\kappa \in \mathfrak{C}(\sim)$, let $C_\kappa \subseteq \kappa$ be the of $\preceq$-maximal elements in κ. Let $M_\kappa = \kappa \setminus C_\kappa$. Then we have $D = \{M_\kappa \mid \kappa \in \mathfrak{C}(\sim)\}$ and $\mu^{\sim}_{\text{Sch}}(M_\kappa) = C_\kappa$.

We call a representative merge of $\mathcal{A}$ w.r.t. $\mu^{\sim}_{\text{Sch}}$ a *Schewe automaton*.

The idea behind the Schewe merger is that whenever an equivalence class of $\sim$ is reached, the transition is redirected to another element of the same equivalence class that lies as "deep" in the automaton as possible.

Lemma 2.2.1. *For a given $\mathcal{A}$ and $\sim$, $\mu^{\sim}_{\text{Sch}}$ can be computed in $\mathcal{O}(|\mathcal{A}|)$.*

Proof. As seen in Lemma 2.1.15, $\preceq$ 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. $\square$

Now that we have established the definition and possible computation of the Schewe merger function, 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 $\mathcal{S} = (Q_{\mathcal{S}}, \Sigma, \delta_{\mathcal{S}}, c_{\mathcal{S}})$ as a Schewe automaton with $\mu^{\sim}_{\text{Sch}}$.

Lemma 2.2.2. *Let ρ be a run on α in $\mathcal{S}$. Then for all i, $\rho(i) \preceq \rho(i+1)$. Furthermore, we have $\rho(i) \prec \rho(i+1)$ if and only if $\rho(i) \prec r_{[\delta(\rho(i),\alpha(i))]_\sim}$.*

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 $\rho(i) \prec r_{[\delta_{\mathcal{A}}(\rho(i),\alpha(i))]_\sim}$, then the transition is redirected to $\rho(i+1) = r_{[\delta_{\mathcal{A}}(\rho(i),\alpha(i))]_\sim}$ and the statement holds.

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

Lemma 2.2.3. *Let $p, q \in Q_{\mathcal{S}}$. 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 $\mathcal{S}$ that reaches p. By Lemma 2.2.2, 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 2.2.4. *Let $\rho \in Q^\omega$ be an infinite run in $\mathcal{S}$ starting at a reachable state. 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 2.2.2, 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 2.2.5. *Let $\sim \subseteq \equiv_L$. Then $\mathcal{A}$ and $\mathcal{S}$ are language equivalent.*

Proof. Let $\alpha \in \Sigma^\omega$ be a word and let ρ be of $\mathcal{S}$ starting in q_0 on α. By Lemma 2.2.4, 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 $\alpha \in L(\mathcal{S})$ 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 $\mathcal{S}$ iff it is accepting in $\mathcal{A}$.

Let $w \in \Sigma^*$ be the prefix of α with $\alpha = w\beta$. By Lemma 2.1.12, we know that $\delta^*(q_0, w) \sim \delta^*_{\mathcal{S}}(q_0, w)$. Since every state is $\sim$-equivalent to its representative and $\sim$ is a congruence relation, we also know $\delta^*_{\mathcal{S}}(q_0, w) \sim \delta^*_{\mathcal{S}}(r_{[q_0]_\sim}, w)$. From $\delta^*_{\mathcal{S}}(r_{[q_0]_\sim}, w)$, the run π accepts β iff $\alpha \in L(\mathcal{S})$. As $\sim$ implies language equivalence, the same must hold for $\delta^*_{\mathcal{A}}(q_0, w)$. Therefore, $\alpha \in L(\mathcal{A})$ iff $\alpha \in L(\mathcal{S})$. □

We have proven that the Schewe merger function can be used to refine congruence relations (such as $\equiv_{\text{Ankh}}$) that, by themselves are not strong enough criteria to allow for a merging of states, to the point that they can be used for state space reduction. A final result regarding this technique is adapted from [12] and shows a relation between this algorithm and priority almost equivalence.

Lemma 2.2.6. *Let $\sim = \equiv_{Ankh}$ and let $\mathcal{S}'$ be a representative merge of $\mathcal{S}$ w.r.t. $\equiv_M$. 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 construct a word on which their priorities differ infinitely often.

First of all, $\mathcal{B}$ must have "the same" $\sim$-equivalence classes as $\mathcal{S}'$, i.e. for all states p in $\mathcal{S}'$, there is a state q in $\mathcal{B}$ s.t. $\mathcal{B}_q$ and $\mathcal{S}'_p$ are priority almost-equivalent. If this would not be the case, there is a p for which no such q exists. As $\mathcal{S}'$ was minimized, p is reachable by some word w. Whatever state q is reached in $\mathcal{B}$ via w, $\mathcal{B}_q$ and $\mathcal{S}'_p$ are not priority almost-equivalent and there is some witness α for that. Hence, $w\alpha$ is a witness that $\mathcal{B}$ and $\mathcal{S}'$ are not priority almost-equivalent.

We define f as a function that maps every $\sim$-equivalence class in $\mathcal{S}'$ to its respective class in $\mathcal{B}$. $\mathcal{B}$ has at least as many $\sim$-equivalence classes but less states than $\mathcal{S}'$, so there is a class $\mathfrak{c}$ in $\mathcal{S}'$ such that $f(\mathfrak{c})$ contains less elements than $\mathfrak{c}$.

By Lemma 2.2.3, there is a unique SCC C in $\mathcal{S}'$ in which all states in $\mathfrak{c}$ are contained. The same must be true for some SCC D in $\mathcal{B}$ for the states in $f(\mathfrak{c})$. Otherwise we could apply the

Schewe automaton construction to $\mathcal{B}$ which does not increase the number of states, is priority almost-equivalent to $\mathcal{B}$, and has this property.

C and D must be non-trivial SCCs. If C would be trivial, then $\mathfrak{c}$ would only contain one element and $f(\mathfrak{c})$ would be empty. Hence, one can force multiple visits to a state in $\mathfrak{c}$ in $\mathcal{S}'$. If D would be trivial, this would not be possible and we could again find a seperating witness of the two automata.

We claim: There is a state $p \in \mathfrak{c}$ s.t. for all $q \in f(\mathfrak{c})$, there is a word $w_q \in \Sigma^*$ that is a witness for non-Moore equivalence of $\mathcal{B}_q$ and $\mathcal{S}'_r$ and $\mathcal{S}'$ does not leave C when reading w_q from p. Assume the opposite, i.e. for all $p \in \mathfrak{c}$, there is a $q_p \in f(\mathfrak{c})$ which does not satisfy said property.

As $|\mathfrak{c}| < |f(\mathfrak{c})|$, there are two states p_1 and p_2 such that $q_{p_1} = q_{p_2} =: q$. For both $i \in \{1, 2\}$ and for each word $w \in \Sigma^*$, we have $\lambda_{\mathcal{B}}(q, w) = \lambda_{\mathcal{S}'}(p_i, w)$ or $\mathcal{S}'$ leaves the SCC C when reading w from p_i. If for both i and all words w the first case would apply, then p_1 and p_2 would be Moore-equivalent and would have been merged in the minimization process of $\mathcal{S}'$. Hence, there are i (which we assume to be $i = 1$ wlog) and w such that $\lambda_{\mathcal{B}}(q, w) \neq \lambda_{\mathcal{S}'}(p_i, w)$ but $\mathcal{S}'_{p_i}$ leaves C when reading w. Let this w be minimal in length.

Let ρ and π be the runs of $\mathcal{S}'$ from p_1 and p_2 via w. At some position k, ρ leaves C. By Lemma 2.2.2, this means that a redirected transition was taken to $\rho(k+1) = r_{[\rho(k+1)]_\sim}$. $\sim$ is a congruence relation and $p_1 \sim p_2$, so $\pi(k+1) \sim \rho(k+1)$. As $p_1 \simeq p_2$ and $p_1 \prec r_{[\rho(k+1)]_\sim}$, we also have $p_2 \prec r_{[\rho(k+1)]_\sim}$ and $\pi(k+1) = \rho(k+1)$. As w was chosen minimal in length, the priorities of the runs are equal everywhere except for $\delta^*_{\mathcal{S}'}(p_1, w)$ and $\delta^*_{\mathcal{S}'}(p_2, w)$. However, the runs converge at $k+1$ which means that they visit the same states from that point on. In particular, $\delta^*_{\mathcal{S}'}(p_1, w) = \delta^*_{\mathcal{S}'}(p_2, w)$.

We have thus proven that the described state $p \in \mathfrak{c}$ and the words $w_q \in \Sigma^*$ exist. We can use these to finally construct our witness α. We define a sequence $(\alpha_n)_{n\in\mathbb{N}} \in \Sigma^*$ such that on α_n, the runs of $\mathcal{S}'$ and $\mathcal{B}$ differ at least n times in priority. Then $\alpha := \bigcup_n \alpha_n$ satisfies our requirements. Furthermore, we make sure that after reading α_n, $\mathcal{S}'$ always stops in p.

Every state in $\mathcal{S}'$ is reachable, so let α_0 be a word that reaches p from the initial state. Now assume that α_n was already defined. Let $q = \delta^*_{\mathcal{B}}(q_0^{\mathcal{B}}, \alpha_n)$. As $p \in \mathfrak{c}$, we can use the same argument as earlier to find $q \in f(\mathfrak{c})$. There is a suitable word w_q that is a witness for Moore non-equivalence while staying in C. As we stay in C, there is also a word u such that $\delta^*_{\mathcal{S}'}(p, w_q u) = p$. We set $\alpha_{n+1} := \alpha_n w_q u$. By choice of w_q, there is a position during the segment on $w_q u$ at which runs of $\mathcal{S}'$ and $\mathcal{B}$ differ in priority. By induction, α_{n+1} satisfies all our required properties. $\square$

Bibliography

[1] Filippo Bonchi and Damien Pous. Checking nfa equivalence with bisimulations up to congruence. In *Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '13, pages 457–468, New York, NY, USA, 2013. ACM.

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

[3] 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.

[4] 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.

[5] 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.

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

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

[8] 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.

[9] 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.

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

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

[12] 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.

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