Automata and Formal Languages

Lecture notes WS 2013/2014

Manfred Kufleitner
kufleitn@in.tum.de

January 28, 2014

Contents

Part I.

Finite Words

1. Rational Sets

Let M be a monoid. The *rational subsets* of M are denoted by $\mathrm{RAT}(M)$. Their inductive definition is as follows.

- Every finite subset of M is in $\mathrm{RAT}(M)$.
- If $K, L \in \mathrm{RAT}(M)$, then $K \cup L \in \mathrm{RAT}(M)$ and $KL \in \mathrm{RAT}(M)$.
- If $K, L \in \mathrm{RAT}(M)$, then $KL \in \mathrm{RAT}(M)$.
- If $L \in \mathrm{RAT}(M)$, then $L^* \in \mathrm{RAT}(M)$.

This means that $\mathrm{RAT}(M)$ is the closure of the finite subsets of M under union, product, and Kleene star. Note that $\emptyset^* = \{1\}$.

Example 1.1. Consider the monoid $M = \mathbb{N} \times \mathbb{N} \times \mathbb{N}$ with componentwise addition. The subset $L = \{\, (\ell, m, n) \mid 2(\ell + m) = n,\ m \geq 2 \,\} \subseteq M$ is rational since

$$L = \{(0,2,4)\}\,\{(0,1,2),(1,0,2)\}^* .$$

We note that if the operation in M is addition, then the product KL for $K, L \subseteq M$ is sometimes denoted by $K{+}L$. In the above case, L could also be written as $\{(0,2,4)\}{+}\{(0,1,2),(1,0,2)\}^*$. ◇

Proposition 1.1. *Let M be a monoid. Then every rational subset of M is contained in a finitely generated submonoid of M.*

Proof. The proof is by induction on the number of operations used for defining $L \in \mathrm{RAT}(M)$. If $L = \{u_1, \ldots, u_n\}$, then L is contained in $\{u_1, \ldots, u_n\}^*$. Let now $K \subseteq \{a_1, \ldots, a_m\}^*$ and $L \subseteq \{b_1, \ldots, b_n\}^*$. Then both $K \cup L$ and KL are contained in $\{a_1, \ldots, a_m, b_1, \ldots, b_n\}^*$, and L^* is contained in $\{b_1, \ldots, b_n\}^*$. □

Example 1.2. If Σ is an infinite set, then the free monoid $M = \Sigma^*$ is not finitely generated. By Proposition 1.1 we see that in this case we have $M \notin \mathrm{RAT}(M)$. More generally, a monoid M is finitely generated if and only if $M \in \mathrm{RAT}(M)$. ◇

1.1. Rational expressions

Let Σ be an arbitrary set. The *rational expressions* over Σ are purely syntactic object. Their definition is as follows:

- The empty set $\emptyset$ and the symbols $a \in \Sigma$ are rational expressions over Σ.
- If s and t are rational expressions over Σ, then so is $(s \mid t)$.
- If s and t are rational expressions over Σ, then so is (st).
- If s is a rational expressions over Σ, then so is s^*.

The *size* of a rational expression is its length as a word over the alphabet Σ together with the symbols '$\emptyset$', '$\mid$', backets '(' and ')', and the star operator '$*$'. The operator '$\mid$' is called *choice*. Whenever possible, the bracketing is omitted and the usual order of operations is applied (star before concatenation, before choice).

Remark 1.1. The choice operator is often written as $+$. We use $\mid$ in order to avoid confusion with the operation in commutative monoids. ◇

If Σ is a subset of a monoid M, then every rational expressions s defines a subset $L(s) \subseteq M$. For rational expressions s and t we set

$$\begin{aligned} L(\emptyset) &= \emptyset \\ L(a) &= \{a\} \\ L(s \mid t) &= L(s) \cup L(t) \\ L(st) &= L(s)L(t) \\ L(s^*) &= L(s)^*, \end{aligned}$$

where $L(s)$ and $L(t)$ are defined inductively. By definition, rational expressions only define rational sets. If Σ is a generating set, then the converse also holds. In particular, rational expressions over Σ can be used as finite presentations of all rational sets.

Proposition 1.2. *Let M be a monoid and let Σ be a generating set. For every subset $L \subseteq M$ the following conditions are equivalent:*

(a) $L \in \mathrm{RAT}(M)$.

(b) *There exists a rational expression s over Σ such that $L(s) = L$.*

Proof. (a) ⇒ (b): Since union (product, Kleene star, respectively) is covered by the operation choice (concatenation, star, respectively), it suffices to show that there exists a rational expression for every finite set. We have $\emptyset = L(\emptyset)$. Thus by closure under union it suffices to show that every singleton set $\{u\}$ can be defined by a rational expression. Let $u = a_1 \cdots a_n$ for $n \geq 0$ and $a_i \in \Sigma$. We proceed by induction over n. If $n = 0$, then $u = 1$ and we have $\{1\} = \emptyset^* = L(\emptyset^*)$. Let now $n > 0$. By induction there exists a rational expression s with $L(s) = \{a_1 \cdots a_{n-1}\}$. Then sa_n is the desired rational expression for $\{u\}$.

(b) ⇒ (a): Both $\emptyset$ and $\{a\}$ are finite sets, and $\mathrm{RAT}(M)$ is closed under union, product, and Kleene star. Thus every rational expression over Σ defines a rational subset of M. □

The set $\{1\}$ containing only the neutral element is defined by the regular expression $\emptyset^*$. Frequently, one allows an additional symbol 1 in rational expressions as a shortcut for $\emptyset^*$, *i.e.*, the semantics of the rational expression 1 is $L(1) = \{1\}$. The idea is that over free monoids, ε is the neutral element in which case we have $L(1) = \{\varepsilon\}$.

1.2. Closure properties of rational sets

By definition, rational sets are closed under union, product, and Kleene star. The following proposition shows that the homomorphic image of a rational set is again rational.

Proposition 1.3. *Let M and N be monoids, and let $\psi : M \to N$ be a homomorphism. If $L \in \mathrm{RAT}(M)$, then $\psi(L) \in \mathrm{RAT}(N)$. Moreover, if ψ is surjective, then for every $K \in \mathrm{RAT}(N)$ there exists $L \in \mathrm{RAT}(M)$ with $\psi(L) = K$.*

Proof. By Proposition 1.2 it suffices to show that for every rational expression s over M there exists a rational expression $\hat{s}$ over N with $L(\hat{s}) = \psi(L(s))$. We set $\hat{\emptyset} = \emptyset$ and $\hat{a} = \psi(a)$ for $a \in M$. For choice, concatenation and star we define

$$\begin{aligned} \widehat{s \mid t} &= \hat{s} \mid \hat{t} \\ \widehat{st} &= \hat{s}\hat{t} \\ \widehat{s^*} &= \hat{s}^*. \end{aligned}$$

The atomic expressions s satisfy $L(\hat{s}) = \psi(L(s))$. For the other operations, induction on the size yields

$$\begin{aligned} L(\widehat{s \mid t}) = L(\hat{s} \mid \hat{t}) = L(\hat{s}) \cup L(\hat{t}) = \psi\big(L(s)\big) \cup \psi\big(L(t)\big) = \psi\big(L(s) \cup L(t)\big) = \psi\big(L(s \mid t)\big) \\ L(\widehat{st}) = L(\hat{s}\hat{t}) = L(\hat{s})L(\hat{t}) = \psi\big(L(s)\big)\,\psi\big(L(t)\big) = \psi\big(L(s)L(t)\big) = \psi\big(L(st)\big) \\ L(\widehat{s^*}) = L(\hat{s}^*) = L(\hat{s})^* = \psi\big(L(s)\big)^* = \psi\big(L(s)^*\big) = \psi\big(L(s^*)\big). \end{aligned}$$

This shows that a rational expression for $\psi(L(s))$ can be obtained by replacing the atomic terms by their images under ψ and not changing the rest of the expression.

For the second part of the statement, when given a rational expression s over N, then we replace every atom $b \in N$ by some arbitrary preimage $\tilde{b} \in \psi^{-1}(b)$. This yields an expression $\tilde{s}$ with $\psi(L(\tilde{s})) = L(s)$. □

Note that if the homomorphism $\psi : M \to N$ is surjective and $K \in \mathrm{RAT}(N)$, then Proposition 1.3 does not yield an expression for $\psi^{-1}(K)$; it just constructs some rational preimage of K. The following example shows that $\psi^{-1}(K)$ might be not rational.

Example 1.3. If M is not finitely generated, then by Proposition 1.1 we have $M \notin \mathrm{RAT}(M)$. When considering the trivial homomorphism $\psi : M \to \{1\}$ with $\psi(u) = 1$ for all $u \in M$, then $M = \psi^{-1}(\{1\})$ is the inverse homomorphic image of the rational set $\{1\}$. In particular, rational sets are not closed under inverse homomorphisms. ◇

Example 1.4. Let Σ be an infinite set and let $M = \Sigma^* \cup \{0\}$ be the free monoid over Σ together with a zero element 0, *i.e.*, we have $u0 = 0u = 0$ for all $u \in M$. Sets of the form $u^{-1}L = \{\, v \in M \mid uv \in L \,\}$ and $Lu^{-1} = \{\, v \in M \mid vu \in L \,\}$ for some $u \in M$ are called *residuals* of $L \subseteq M$. We have $\{0\} \in \mathrm{RAT}(M)$ whereas the residual $0^{-1}\{0\} = M$ is not rational. This shows that rational sets are not closed under residuals. ◇

We will see in Section 3.4 that rational sets are neither closed under intersection and nor under complement.

1.3. Nondeterministic automata

Automata are a simple machine model for defining sets. They come in two flavors, deterministic and nondeterministic. In general, finite nondeterministic automata can define more sets than finite deterministic automata. It will turn out that finite nondeterministic automata can accept exactly the rational sets. A *nondeterministic M-automaton* $\mathcal{A} = (Q, \delta, I, F)$ consists of

- a set of *states* Q,
- a *transition relation* $\delta \subseteq Q \times M \times Q$,
- a set of *initial states* I,
- and a set of *final states* F.

The elements of δ are called *transitions.* If (p, u, q) is a transition, then u is called its *label.* The notion of a run is the main tool for defining the set accepted by $\mathcal{A}$. A *run* of the automaton $\mathcal{A}$ is a sequence

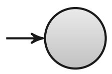
$$r = q_1 u_1 q_2 \cdots u_n q_{n+1}$$

with $(q_i, u_i, q_i) \in \delta$ for all $1 \leq i \leq n$. We say that r is a run from q_1 to q_{n+1} on the element $u = u_1 \cdots u_n$. If $n = 0$, then r is a run on the neutral element. One way of how to think about runs on $u \in M$ is that the automaton $\mathcal{A}$ decorates a factorization of u with states such that locally the transition relation holds. The run r is *accepting* if $q_1 \in I$ and $q_{n+1} \in F$. The subset of M *accepted* by $\mathcal{A}$ is

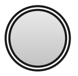
$$L(\mathcal{A}) = \{\, u \in M \mid \text{there exists an accepting run of } \mathcal{A} \text{ on } u \,\}.$$

We write $p \xrightarrow{u} q$ if there exists a run from state p to state q on the element $u \in M$. Using this notation, we have $L(\mathcal{A}) = \left\{ u \in M \;\middle|\; \exists i \in I \, \exists f \in F \colon i \xrightarrow{u} f \right\}$. If $p \xrightarrow{u} q$ and $q \xrightarrow{v} q'$, then $p \xrightarrow{uv} q'$.

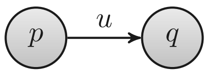
Automata can be visualized as edge-labeled graphs. The states of the automaton are the vertices and transition define the edges. Moreover, initial states are marked by an ingoing arrow (from nowhere), and final states have a double circle. Transitions of the form (q, u, q) are called *loops*.



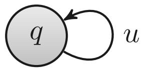
initial state



final state



transition (p, u, q)

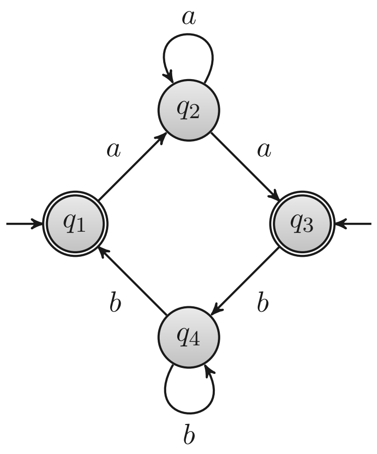


loop (q, u, q)

Example 1.5. Let $\Sigma = \{a, b\}$. Consider the nondeterministic Σ^*-automaton $\mathcal{A} = (Q, \delta, I, F)$ with states $Q = \{ q_1, q_2, q_3, q_4 \}$, initial and final states $I = F = \{q_1, q_3\}$ and transition relation

$$\delta = \{ (q_1, a, q_2), \, (q_2, a, q_2), \, (q_2, a, q_3), \, (q_3, b, q_4), \, (q_4, b, q_4), \, (q_4, b, q_1) \} \, .$$

The automaton $\mathcal{A}$ can be visualized as follows:



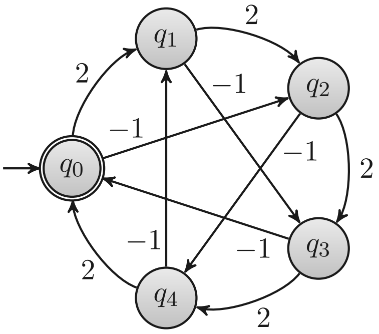
We will see that it is not suprising that $L(\mathcal{A}) = L((aaa^* \mid bbb^*)^*)$ can be written as a rational expression. We have $q_2 \xrightarrow{aaa} q_2$ and $q_2 \xrightarrow{aaa} q_3$. An accepting run of $\mathcal{A}$ on $aaabbbaa$ is

$$q_1 \, a \, q_2 \, a \, q_2 \, a \, q_3 \, b \, q_4 \, b \, q_4 \, b \, q_1 \, a \, q_2 \, a \, q_3 \, .$$

Runs can be obtained from the visualization of an automaton by chasing paths. ◇

Two nondeterministic M-automata are *equivalent* if they accept the same subset. A nondeterministic M-automaton is *deterministic* if $|I| = 1$ and if for every state $p \in Q$ and every element $u \in M$ there exists a unique state $q \in Q$ with $p \xrightarrow{u} q$. This means for every state $p \in Q$ and every element $u \in M$, all runs on u starting at p end in a unique state q. This does not mean that there is only one run on u from p to q.

Example 1.6. We consider the integers $\mathbb{Z}$ with addition. The following nondeterministic $\mathbb{Z}$-automaton is deterministic.



It accepts the subset $5\mathbb{Z} = \{\, 5k \mid k \in \mathbb{Z} \,\}$. The following are some of the accepting runs of $0 \in \mathbb{Z}$:

$$q_0$$
$$q_0\, 2\, q_1\, (-1)\, q_3\, (-1)\, q_0$$
$$q_0\, (-1)\, q_2\, 2\, q_3\, (-1)\, q_0$$
$$q_0\, 2\, q_1\, (-1)\, q_3\, 2\, q_4\, (-1)\, q_1\, (-1)\, q_3\, (-1)\, q_0$$

In particular, the accepting run of 0 is not unique. In fact, there are infinitely many of them since every presentation of 0 as a sum of 2's and (-1)'s yields another run. ◇

1.4. Conversion of rational expressions into automata

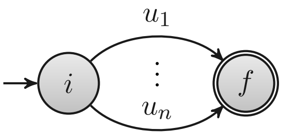
There are many ways for translating rational expressions into automata. The one we use here is due to Kenneth Lane Thompson who is well-known for his contributions to the unix operating system. This method is therefore called the *Thompson construction.*

Theorem 1.4. *Let M be a monoid. For every rational expression s over M there exists a nondeterministic M-automaton $\mathcal{A}_s$ with $\mathcal{O}(|s|)$ transitions such that $L(\mathcal{A}_s) = L(s)$.*

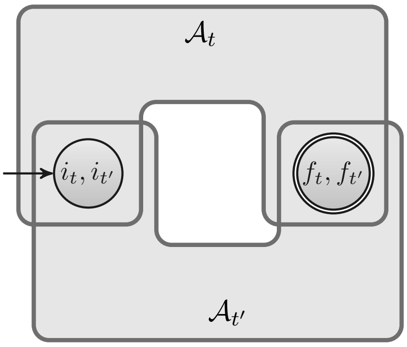
Proof. Let $1 \in M$ be the neutral element. For every rational expression s we construct an automaton $\mathcal{A}_s$ with the following properties

- $L(\mathcal{A}_s) = L(s)$.
- $\mathcal{A}_s$ has a unique initial state i_s, and i_s has no incoming transitions. *Incoming transitions* of a state q are transitions of the form (p, u, q).
- $\mathcal{A}_s$ has a unique final state f_s, and f_s has no outgoing transitions. *Outgoing transitions* of a state p are transitions of the form (p, u, q).

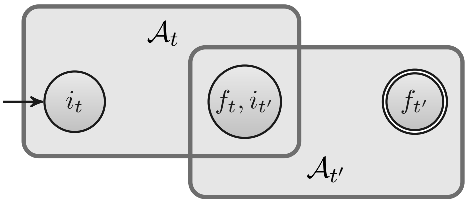
The finite subset $L = \{\, u_1, \ldots, u_n \,\}$ of M is accepted by the following two-state automaton:



This covers the cases $s = \emptyset$, $s = 1$, and $s = a$. For the remaining operations we assume that we have already constructed automata $\mathcal{A}_t$ and $\mathcal{A}_{t'}$. The automaton $\mathcal{A}_s$ for choice $s = t \mid t'$ is:

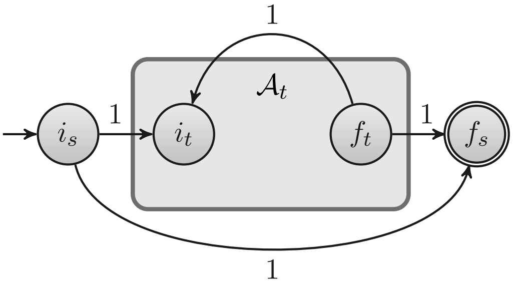


This means, we take the disjoint union of $\mathcal{A}_t$ and $\mathcal{A}_{t'}$, except that we identity i_t with $i_{t'}$ (which yields the initial state i_s of $\mathcal{A}_s$) and f_t with $f_{t'}$ (which yields the final state f_s). The automaton $\mathcal{A}_s$ for concatenation $s = tt'$ is:



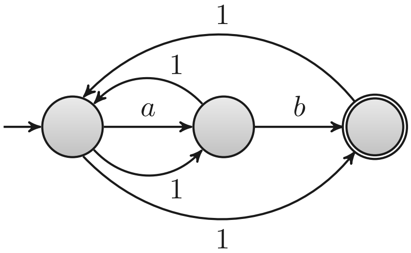
This means we take the union of $\mathcal{A}_t$ and $\mathcal{A}_{t'}$ in which we identify the states f_t and $i_{t'}$, other than that the union is disjoint. The initial state of $\mathcal{A}_s$ is i_t and the final state is $f_{t'}$.

For $s = t^*$ the construction is as follows:



In all cases we have $L(\mathcal{A}_s) = L(s)$. □

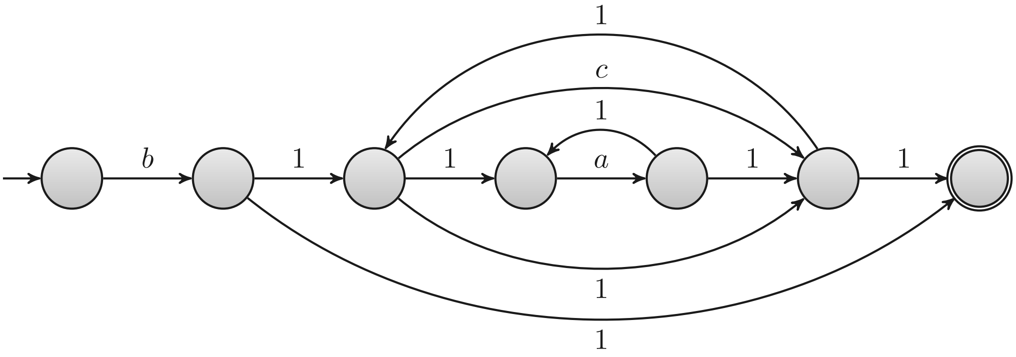
Remark 1.2. It is tempting to optimize the construction for the star operator by identifying i_s with i_t and f_s with f_t. This destroys the invariant that i_s has no incoming transitions and f_s has no outgoing transitions. But without this presumption, none of the constructions for choice, concatenation, and star is correct. For instance, consider monoid $M = \{a, b\}^*$ and the rational expression $s = (a^*b)^*$. The resulting automaton would be



We have $L(s) = \{1\} \cup \{a, b\}^* \, b$ but the above automaton also accepts the word $a \notin L(s)$. ◇

We note that the Thompson construction does not depend on M. This means that for every rational expression s over Σ it constructs a nondeterministic automaton $\mathcal{A}_s$ with labels in $\Sigma \cup \{1\}$ such that over every monoid M with $\Sigma \subseteq M$, the expression s and the automaton $\mathcal{A}_s$ define the same subset.

Example 1.7. Consider the regular expression $s = b(c \,|\, a^*)^*$ over $\Sigma = \{a, b, c\}$. Then the Thompson construction yields the following automaton $\mathcal{A}_s$:



We will use this automaton as a running example. ◇

1.5. Conversion of automata into rational expressions

For translating nondeterministic automata into rational expressions we use a slightly more general automaton model as an intermediate step. An *M-automaton with rational labels* $\mathcal{A} = (Q, \delta, I, F)$ satisfies $\delta \subseteq Q \times \mathrm{RAT}(M) \times Q$. By using singleton sets as labels we obtain the usual nondeterministic M-automata. In order to have finite presentations, the rational labels are frequently given as rational expressions. The automaton $\mathcal{A}$ is finite if δ is finite. Note that if δ is finite, then, for a transition $(p, L, q) \in \delta$, the set L can still be infinite. A sequence $q_1 u_1 q_2 \cdots u_n q_{n+1}$ is a *run* of $\mathcal{A}$ if for every $i \in \{1, \ldots, n\}$ there exists a transition $(q_i, L_i, q_{i+1}) \in \delta$ with $u_i \in L_i$. The following procedure for translating automata into rational expressions is called *state elimination*.

Theorem 1.5. *Let M be a monoid. For every finite M-automaton $\mathcal{A}$ with rational labels we have $L(\mathcal{A}) \in \mathrm{RAT}(M)$.*

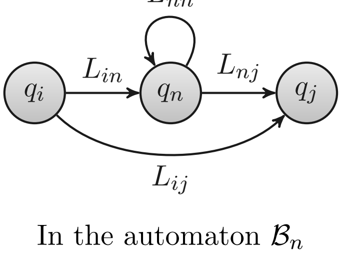
Proof. Let $\mathcal{A} = (Q, \delta, I, F)$ with $Q = \{q_2, \ldots, q_n\}$. Since rational sets are closed under union, we can assume that for any pair of states q_i, q_j there exists at most one transition $(q_i, L_{ij}, q_j) \in \delta$. If there is no transition (q_i, L_{ij}, q_j), then we set $L_{ij} = \emptyset$. We introduce a new unique initial state q_0 and a new unique final state q_1 by setting $Q_n = \{\, q_0, \ldots, q_n \,\} = Q \cup \{q_0, q_1\}$ and

$$\delta_n = \delta \cup \{\, (q_0, \{1\}, i) \mid i \in I \,\} \cup \{\, (f, \{1\}, q_1) \mid f \in F \,\}.$$

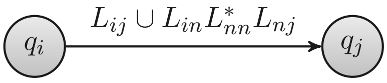
The automaton $\mathcal{B}_n = (Q_n, \delta_n, \{q_0\}, \{q_1\})$ is equivalent to $\mathcal{A}$. Note that q_0 has no incoming transitions and that q_1 has no outgoing transitions. If $n > 1$, then we construct an equivalent automaton $\mathcal{B}_{n-1} = (Q_{n-1}, \delta_{n-1}, \{q_0\}, \{q_1\})$ with n states as follows. We set $Q_{n-1} = \{q_0, \ldots, q_{n-1}\}$ and

$$\delta_{n-1} = \{\, (q_i, L_{ij} \cup L_{in} L_{nn}^* L_{nj}, q_j) \mid 0 \le i, j < n \,\}$$

This means that we replace every transition $(q_i, L_{ij}, q_j) \in \delta_n$ by $(q_i, L_{ij} \cup L_{in} L_{nn}^* L_{nj}, q_j)$. This can be visualized as follows:



In the automaton $\mathcal{B}_n$

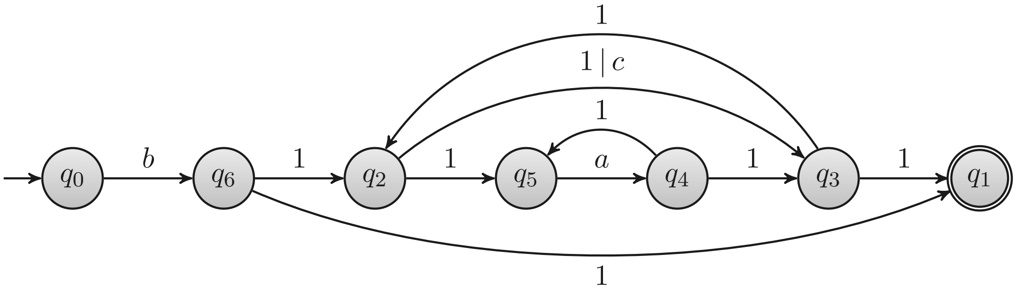


In the automaton $\mathcal{B}_{n-1}$

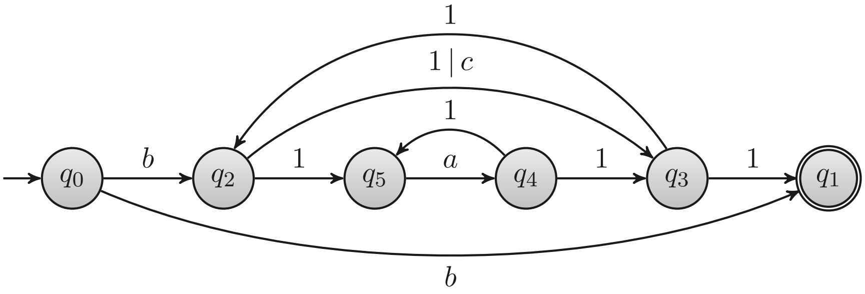
All labels in the automaton $\mathcal{B}_{n-1}$ are rational, and $\mathcal{B}_{n-1}$ is equivalent to $\mathcal{B}_n$. Note that all incoming transitions of q_0 and all outgoing transitions of q_1 have label $\emptyset$. By repeated state elimination we obtain the equivalent automaton $\mathcal{B}_1$. If $(q_0, L_{01}, q_1) \in \delta_1$, then $L(\mathcal{B}_1) = L_{01} \in \mathrm{RAT}(M)$; otherwise we have $L(\mathcal{B}_1) = \emptyset \in \mathrm{RAT}(M)$. □

Remember $\emptyset^* = \{1\}$. Note that when giving all labels L_{ij} in the state elimination algorithm as rational expressions s_{ij}, then the above construction does not depend on M. If $\mathcal{A}$ is a nondeterministic M-automaton with n states and labels in Σ, then the state elimination algorithm constructs a rational expression of size $|\Sigma|\, 2^{\mathcal{O}(n)}$. At the beginning, each language L_{ij} has size at most $|\Sigma|\, n$. Then while eliminating one state, the size of the longest expression at most quadruples (plus some constant for choice, star, and brackets). This yields $|\Sigma|\, n 2^{\mathcal{O}(n)} = |\Sigma|\, 2^{\mathcal{O}(n)}$ as an upper bound on the size.

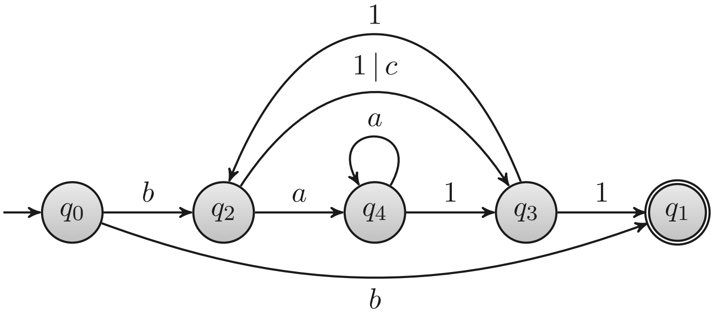
Example 1.8. Consider the following automaton $\mathcal{A}$ from Example 1.7.



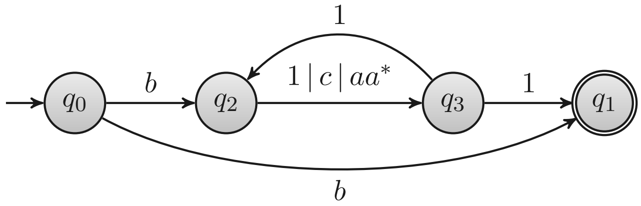
Since q_0 has no incoming transitions and q_1 has no outgoing transitions, we do not have to add additions states. We always apply obvious simplifications involving concatenation with 1. After elimination of q_6 we obtain:



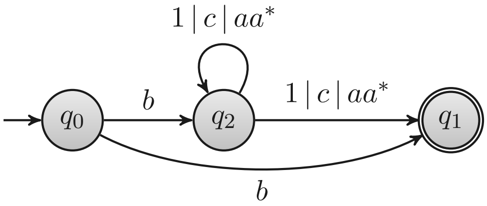
Eliminating q_5 yields:



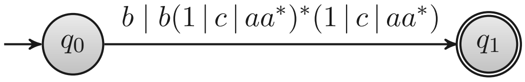
After elimination of q_4 we have:



By elimination of q_3 we obtain:



Finally eliminating q_2 yields:



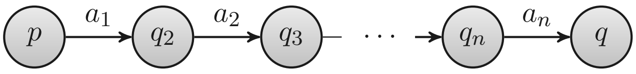
Therefore, a rational expression for $L(\mathcal{A})$ is $b \mid b(1 \mid c \mid aa^*)^*(1 \mid c \mid aa^*)$. Remember that we constructed $\mathcal{A}$ from the expression $b(c \mid a^*)^*$ using the Thompson construction. ◇

1.6. Removal of epsilon-transitions

Let Σ be a generating set of the monoid M and let $\mathcal{A} = (Q, \delta, I, F)$ be a nondeterministic M-automaton. We say that $\mathcal{A}$ is a *letter-by-letter automaton for* Σ if all labels are in Σ, *i.e.*, the transition relation satisfies $\delta \subseteq Q \times \Sigma \times Q$. Every transition with label $u = a_1 \cdots a_n$ with $n \geq 1$ and $a_i \in \Sigma$ can easily be split into n transitions. Avoiding transitions with label $u = 1$ is slightly more involved. Since ε is the neutral element of the free monoid, the following procedure is called *removal of ε-transitions*.

Theorem 1.6. *Let M be a monoid with generating set Σ. For every finite nondeterministic M-automaton $\mathcal{A}$ there exists an equivalent finite letter-by-letter automaton for Σ with only one initial state.*

Proof. Let $\mathcal{A} = (Q, \delta, I, F)$. By omitting all unreachable states we can assume that Q is finite. As long as there are transitions (p, u, q) with label $u \in M \setminus (\Sigma \cup \{1\})$ we write $u = a_1 \cdots a_n$ with $a_i \in \Sigma$ and we replace (p, u, q) by



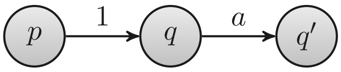
where $q_2, \ldots, q_n$ are new states. After this step, all labels are in $\Sigma \cup \{1\}$ and the accepted set did not change. Let q_0 be a new state. We add the transitions $\{\, (q_0, 1, i) \mid i \in I \,\}$ and we choose q_0 as the only initial state. Next, we make the 1-transitions transitive, *i.e.*, whenever there are two consecutive transitions $(p, 1, q)$ and $(q, 1, r)$, then we add the transition $(p, 1, r)$. One can think of this procedure as introducing shortcuts. Let

$$F' = F \cup \{\, p \in Q \mid (p, 1, q) \text{ is a transition with } q \in F \,\}.$$

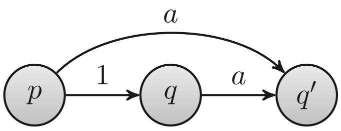
Using F' as final sets yields the equivalent automaton $\mathcal{B}' = (Q', \delta', \{q_0\}, F')$. Next, for every transition $(p, 1, q) \in \delta'$ and every generator $a \in \Sigma$ we add the transitions

$$\{\, (p, a, q') \mid (q, a, q') \in \delta' \,\}.$$

This defines the equivalent automaton $\mathcal{B}'' = (Q', \delta'', \{q_0\}, F')$.



In the automaton $\mathcal{B}'$



In the automaton $\mathcal{B}''$

Finally, let $\mathcal{B}$ be obtained from $\mathcal{B}''$ by removing all transitions with label 1. It remains to show $L(\mathcal{B}) = L(\mathcal{B}'')$. The inclusion $L(\mathcal{B}) \subseteq L(\mathcal{B}'')$ is trivial since every accepting run of $\mathcal{B}$ is also an accepting run of $\mathcal{B}''$. Let now $r = q_0 u_1 q_1 \cdots u_n q_n$ be an accepting run of $\mathcal{B}''$ on $u = u_1 \cdots u_n \in L(\mathcal{B}'')$ with as few 1-transitions as possible. We can assume $u \neq 1$ since otherwise $r = q_0$ is an accepting run of $\mathcal{B}$, as desired. By construction of F' we have $u_n \neq 1$. If $u_i = 1$, then $i < n$ and $u_{i+1} \neq 1$ since the 1-transitions are transitive. Now, by construction of δ'' there exists a transition $(q_{i-1}, u_{i+1}, q_{i+1})$. Replacing the segment $q_{i-1} u_i q_i u_{i+1} q_{i+1}$ of the run r by $q_{i-1} u_{i+1} q_{i+1}$ we obtain an accepting run on u with fewer 1-transitions. This is a contradiction to the choice of r. Therefore, r uses no 1-transitions, *i.e.*, r is an accepting run of $\mathcal{B}$. This shows $L(\mathcal{B}'') \subseteq L(\mathcal{B})$. □

We summarize the algorithm for the removal of ε-transitions. Moverover, we give a rough estimate of the running time of each step.

(a) Replace labels $u \in M \setminus (\Sigma \cup \{1\})$ by several transitions with labels in Σ. The running time of this step depends on the number of transitions required for this substitution.

(b) Add a unique initial state. To this end, $|I|$ transitions have to be drawn.

(c) Make the 1-transitions transitive. There are cubic algorithms with respect to the number of states for computing transitive closures (*e.g.* Warshall's algorithm).

(d) All states which can reach a final state using a 1-transition also become final states. Depending on the implementation of the automaton, this step is linear in the number of transitions.

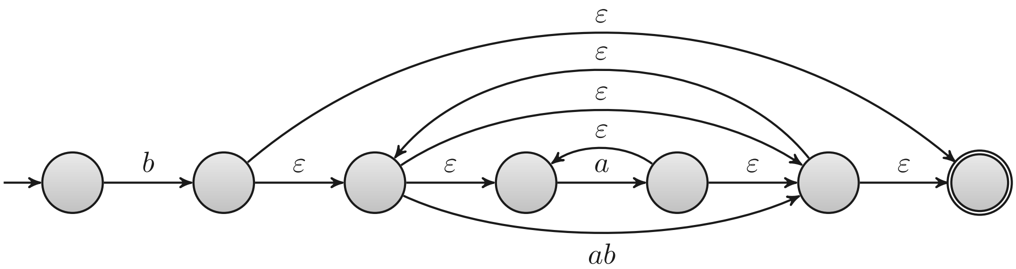
(e) For transitions $(p, 1, q)$ and (q, a, r) with $a \in \Sigma$ we introduce a shortcut (p, a, r). Depending on the implementation of the automaton, this step is quadratic in the number of transitions.

(f) All 1-transitions are removed. This step is linear in the number of transitions.

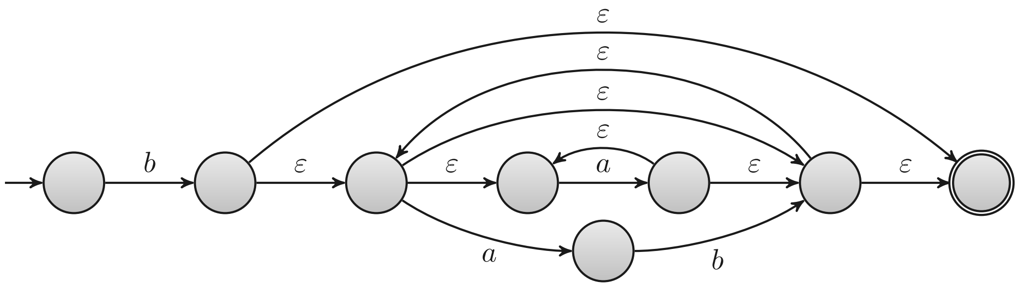
(g) As an optimization, one can omit unreachable states in the resulting automaton.

If all labels are in $\Sigma \cup \{1\}$, then the algorithm can be implemented with a running time of $\mathcal{O}(|Q|^2 \, |\delta|)$. This is mainly due to the complexity of step (e) since an upper bound for the number of 1-transitions after step (c) is $|Q|^2$. Since we can assume that all states are reachable we have $\mathcal{O}(|Q|^3) \subseteq \mathcal{O}(|Q|^2 \, |\delta|)$ which covers the running time of step (c). Also note that if all labels are in $\Sigma \cup \{1\}$, then, except for the initial state, no other new states are required by the construction.

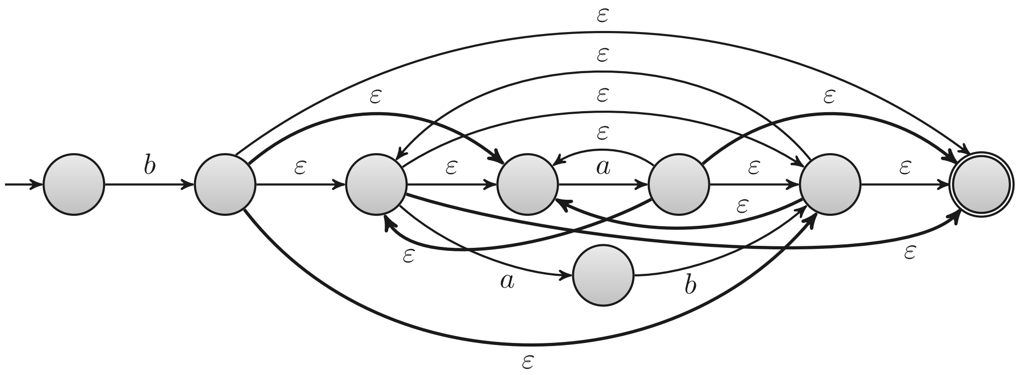
Example 1.9. Let $M = \{a, b\}^*$, let $\Sigma = \{a, b\}$, and consider the following automaton from Example 1.7 in which we replaced c by ab, *i.e.*, the language accepted by the automaton is defined by the rational expression $b(ab \mid a^*)^*$.



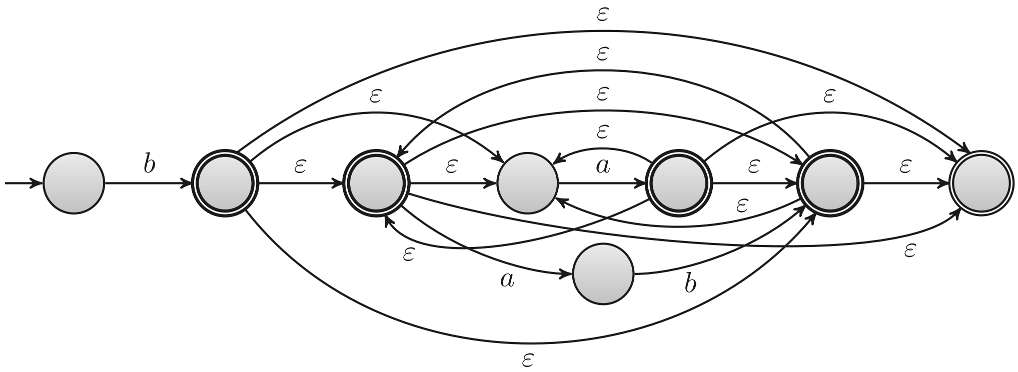
The automaton already has a unique initial state. We proceed by splitting the transition with label ab into two transitions. This yields the following automaton.



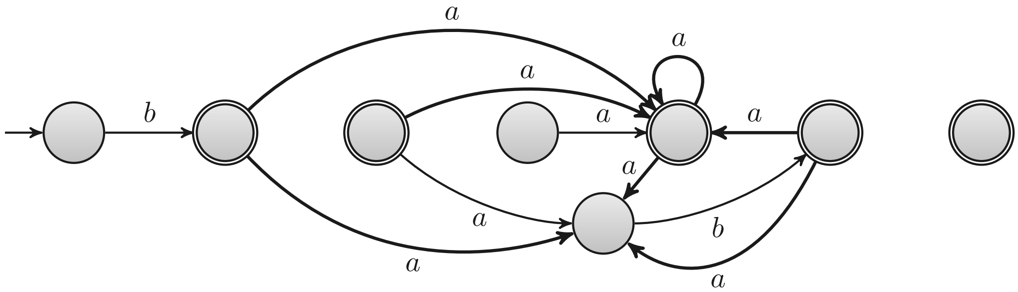
In the next step, we make the ε-transitions transitive by successively adding new ε-transitions as shortcuts for every two consecutive ε-transitions. We thereby omit ε-loops.



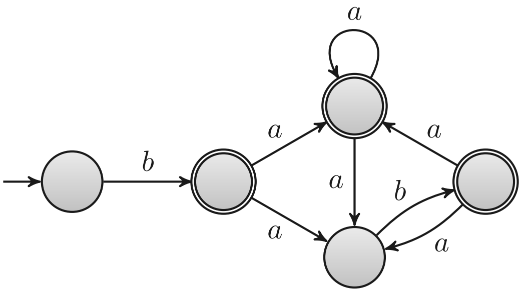
Now, every state which can reach a final state using an ϵ-transition becomes a final state itself.



We combine the next two steps. We introduce new transitions with labels from Σ and we remove all ε-transitions.



Finally, by omitting unreachable states (and after some rearrangement) we obtain the following automaton.



The above automaton is letter-by-letter for $\{a, b\}$ and it has only one initial state. ◇

Remark 1.3. By adapting the algorithm for removal of ε-transitions one can also construct letter-by-letter automata with only one final state; but, in general, these automata then have more than one initial state. Not for every rational set there exists a letter-by-letter automata with only one initial state and only one final state. For instance, let $\Sigma = \{a, b\}$ and $M = \Sigma^*$, and consider the language $L = \{\varepsilon, a, b\}$. Assume there were a letter-by-letter automaton $\mathcal{A}$ with $L(\mathcal{A}) = L$ such that $\mathcal{A}$ has only one initial state q_0 and only one final state q_1. Then, since $\varepsilon \in L$, we have $q_0 = q_1$. From $a, b \in L$ we conclude that there are accepting runs for the words a and b, *i.e.*, there are transitions (q_0, a, q_0) and (q_0, b, q_0). Combining these two runs yields an accepting run for every word in Σ^*. This is a contradiction to $L(\mathcal{A}) = L$. Therefore such an automaton $\mathcal{A}$ for the language L does not exist. ◇

1.7. Equivalence of rational expressions and nondeterministic automaton models

We combine the findings in the previous sections for obtaining the following characterizations of rational sets.

Theorem 1.7. *Let M be a monoid with generating set Σ and let $L \subseteq M$. The following conditions are equivalent:*

(a) *$L \in \mathrm{RAT}(M)$, i.e., the set L is rational.*

(b) *There exists a rational expression s over Σ such that $L(s) = L$.*

(c) *L is accepted by a finite nondeterministic M-automaton.*

(d) *L is accepted by a finite letter-by-letter M-automaton for Σ with a single initial state.*

(e) *L is accepted by a finite M-automaton with rational labels.*

Proof. The equivalence of (a) and (b) is Proposition 1.2. (b) ⇒ (c): This follows from the Thompson construction in Theorem 1.4. (c) ⇒ (d): The removal of ε-transitions in Theorem 1.6 shows this implication. (d) ⇒ (e): Trivial. (e) ⇒ (a): This translation is achieved by state elimination, see Theorem 1.5. □

In Figure 1.1 we summarize the conversions between the different descriptions of rational sets. The numbers after the name of the construction indicate the maximal blow-up in size. For automata we count the number of states. For better readability the given bounds we assume that the set of generators Σ is fixed.

1.8. Semilinear subsets of commutative monoids

A monoid M is *commutative* if $uv = vu$ for all $u, v \in M$. The operation in commutative monoids is usually written as addition. This means we write $u + v$ instead of uv, and

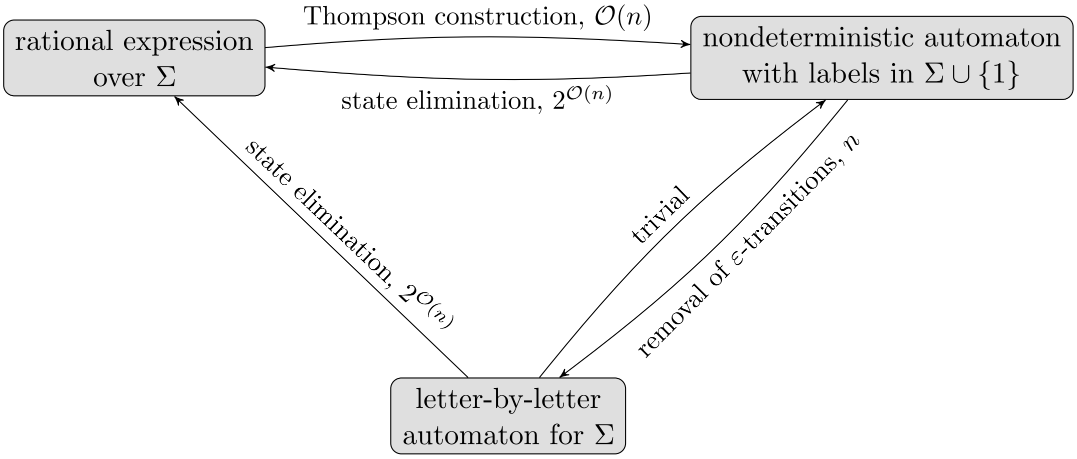

rational expression over Σ
nondeterministic automaton with labels in Σ ∪ {1}
letter-by-letter automaton for Σ
Thompson construction, $\mathcal{O}(n)$
state elimination, $2^{\mathcal{O}(n)}$
state elimination, $2^{\mathcal{O}(n)}$
trivial
removal of ε-transitions, n

Figure 1.1.: Transformation between rational expressions and nondeterministic automata.

the neutral element is denoted by 0. Similarly, for subsets $K, L \subseteq M$ we write $K + L = \{\, u + v \mid u \in K,\ v \in L \,\}$ instead of concatenation KL. Typical commutative monoids are $\mathbb{N}^k$ and $\mathbb{Z}^k$ with componentwise addition. Let M be an arbitrary commutative monoid, let $u \in M$, and let $n \in \mathbb{N}$. Then the term nu denotes the n-fold sum of u, *i.e.*,

$$nu = \underbrace{u + \cdots + u}_{n \text{ terms}}$$

In particular, $0u = 0$ is the neutral element of M. A subset $L \subseteq M$ is *linear* if $L = u_0 + \{u_1, \ldots, u_k\}^*$ for $k \geq 0$ and $u_0, \ldots, u_k \in M$. We have $u \in L$ if and only if there exist $n_1, \ldots, n_k \in \mathbb{N}$ with

$$u = u_0 + n_1 u_1 + \cdots + n_k u_k.$$

A subset $L \subseteq M$ is *semilinear* if L is a finite union of linear sets. Every semilinear set L can be written as $L = \bigcup_{j=1}^{\ell}(v_j + C_j^*)$ for $v_i \in M$ and finite subsets $C_i \subseteq M$. We show that the semilinear and the rational subsets coincide.

Theorem 1.8. *Let M be a commutative monoid and let $L \subseteq M$. The subset L is semilinear if and only if $L \in \mathrm{RAT}(M)$.*

Proof. By definition, every semilinear set is rational. Every finite subset of M is semilinear; and semilinear sets are closed under union. For the remaining inclusion it therefore suffices to show that semilinear sets are closed under addition and Kleene star. Let $K = \bigcup_{i=1}^{k}(u_i + B_i^*)$ and $L = \bigcup_{j=1}^{\ell}(v_j + C_j^*)$ for $u_i, v_j \in M$ and for finite subsets $B_i, C_j \subseteq M$. We have

$$K + L = \bigcup_{i=1}^{k} \bigcup_{j=1}^{\ell} \big((u_i + v_j) + (B_i \cup C_j)^*\big).$$

This shows that $K + L$ is semilinear. For all subsets $X, Y \subseteq M$ we have $(X \cup Y)^* = X^* + Y^*$. It therefore remains to show that the Kleene star of a linear set yields a semilinear set. We have

$$\big(u_0 + \{u_1, \ldots, u_k\}^*\big)^* = \{0\} \cup \big(u_0 + \{u_0, u_1, \ldots, u_k\}^*\big).$$

Thus the Kleene star of the linear set $u_0 + \{u_1, \ldots, u_k\}^*$ is semilinear. □

2. Recognizable Sets

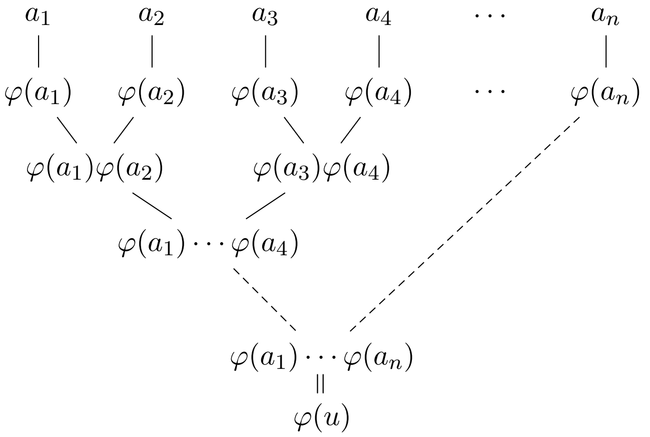
Throughout this chapter, the letters M and N denote monoids. Let $\varphi : M \to N$ be a homomorphism. The homomorphism φ *recognizes* a subset $L \subseteq M$ if $\varphi^{-1}(\varphi(L)) = L$. We also say that a monoid N *recognizes* $L \subseteq M$ if there exists a homomorphism $\varphi : M \to N$ which recognizes L. A subset $L \subseteq M$ is *recognizable* if it is recognized by a finite monoid. The class of all recognizable subsets of M is denoted by $\mathrm{REC}(M)$. For any homomorphism $\varphi : M \to N$ and any subset $L \subseteq M$ we have $L \subseteq \varphi^{-1}(\varphi(L))$ since

$$u \in L \Rightarrow \varphi(u) \in \varphi(L).$$

Thus, if $\varphi^{-1}(\varphi(L)) \subseteq L$, then $\varphi^{-1}(\varphi(L)) = L$ and the following equivalence holds:

$$u \in L \Leftrightarrow \varphi(u) \in \varphi(L).$$

This means that for checking whether or not $u \in L$ holds, it suffices to test $\varphi(u) \in \varphi(L)$; and the latter can be considerably easier. For example, if N is a finite monoid (*i.e.*, if L is recognizable), then this is just a lookup in a finite table. Moreover, the computation of $\varphi(u)$ can be easily parallelized if u is given as a sequence of generators $u = a_1 \cdots a_n$:



With this scheme the parallel computation of $h(u)$ with n processors requires $\log n$ steps, it computes n images of letters under φ, and it evaluates $n - 1$ products in N.

The following lemma shows that recognition can also be defined in terms of accepting sets $P \subseteq N$. In particular, this gives a way for presenting recognizable subsets of M using the following three ingredients: a finite monoid N, a homomorphism $\varphi : M \to N$, and a subset $P \subseteq N$.

Lemma 2.1. *Let $\varphi : M \to N$ be a homomorphism and let $L \subseteq M$. Then φ recognizes L if and only if there exists $P \subseteq N$ with $L = \varphi^{-1}(P)$.*

Proof. For the implication from left to right we can set $P = \varphi(L)$. Let now $L = \varphi^{-1}(P)$. Then $\varphi^{-1}(\varphi(L)) = \varphi^{-1}(\varphi(\varphi^{-1}(P))) \subseteq \varphi^{-1}(P) = L$. □

If $\varphi : M \to N$ is a homomorphism, then $\varphi(M)$ is a submonoid of N. Therefore, by replacing N by $\varphi(M)$ one can assume that φ is surjective. If a surjective homomorphism $\varphi : M \to N$ recognizes $L \subseteq M$, then $P = \varphi(L)$ is the only accepting set for L.

2.1. Closure properties of recognizable sets

If M is finite, then every subset of M is recognizable since it is recognized by the identity mapping $\mathrm{id} : M \to M$. For groups, also the converse statement is true.

Proposition 2.2. *A group G is finite if and only if $\{1\} \in \mathrm{REC}(G)$.*

Proof. It suffices to show the implication from right to left. Let N be a finite monoid and let $\varphi : G \to N$ be a homomorphism recognizing $\{1\}$. We can assume that φ is surjective and hence, N is a group. Suppose $\varphi(u) = \varphi(v)$. Then $\varphi(uv^{-1}) = \varphi(u)\varphi(v)^{-1} = 1$ and thus $uv^{-1} \in \varphi^{-1}(1)$. Since $\varphi^{-1}(1) = \{1\}$ we conclude $uv^{-1} = 1$ and thus $u = v$. Therefore, φ is injective, which shows that G and N are isomorphic. In particular, G is finite. □

If M is finitely generated, then only countably many subsets of M are recognizable. Even though, quite few subsets of infinite monoids are recognizable, this class provides a nice collection of closure properties. Let $L \subseteq M$. A *left residual* of L is a set of the form $u^{-1}L = \{\, v \in M \mid uv \in L \,\}$ for $u \in M$. Symmetrically, a *right residual* of L is $Lu^{-1} = \{\, v \in M \mid vu \in L \,\}$. The following theorem summarizes some important closure properties of recognizable subsets.

Theorem 2.3. *Let M and M' be monoids and let $\psi : M' \to M$ be a homomorphism. The following properties hold:*

(a) $\mathrm{REC}(M)$ *is closed under finite union, finite intersection, and complement.*

(b) $\mathrm{REC}(M)$ *is closed under left and right residuals.*

(c) *If $L \in \mathrm{REC}(M)$, then $\psi^{-1}(L) \in \mathrm{REC}(M')$, i.e., the recognizable sets are closed under inverse homomorphisms.*

Moreover, for effective presentations of M, M' and ψ, all of the above closure properties are effective.

Proof. (a): If a homomorphism φ recognizes $L \subseteq M$, then it also recognizes $M \setminus L$. The trivial monoid $\{1\}$ recognizes both M and $\emptyset$. This covers the empty intersection and the empty union. Consider homomorphism $\varphi_i : M \to N_i$ with $L_i = \varphi_i^{-1}(P_i)$. Let $\varphi : M \to N_1 \times N_2$ with $\varphi(u) = (\varphi_1(u), \varphi_2(u))$. Then $L_1 \cap L_2 = \varphi^{-1}(P_1 \times P_2)$ and $L_1 \cup L_2 = \varphi^{-1}((P_1 \times N_2) \cup (N_1 \times P_2))$.

(b): Let $u \in M$ and let $\varphi : M \to N$ be a homomorphism with $L = \varphi^{-1}(P)$ for $P \subseteq N$. The subset $P' = \varphi(u)^{-1}P = \{\, x \in N \mid \varphi(u)x \in P \,\}$ satisfies

$$\varphi(v) \in P' \Leftrightarrow \varphi(u)\varphi(v) \in P \Leftrightarrow \varphi(uv) \in P \Leftrightarrow uv \in L \Leftrightarrow v \in u^{-1}L.$$

This shows $u^{-1}L = \varphi^{-1}(P')$. Right residuals are symmetric.

(c): Let $K = \psi^{-1}(L)$. If $\varphi : M \to N$ recognizes L, then $\varphi \circ \psi : M' \to N$ recognizes K since

$$u \in K \Leftrightarrow \psi(u) \in L \Leftrightarrow \varphi(\psi(u)) \in \varphi(L).$$ □

We now show that recognizable sets are not closed under homomorphic images.

Example 2.1. Let $\psi : \{a,b\}^* \to \mathbb{Z}$ be the homomorphism defined by $a \mapsto 1$ and $b \mapsto -1$. We have $\{\varepsilon\} \in \mathrm{REC}(\{a,b\}^*)$ but $\psi(\{\varepsilon\}) = \{0\}$ is not in $\mathrm{REC}(\mathbb{Z})$ by Proposition 2.2. ◇

The following example of Shmuel Winograd shows that, in general, recognizable sets are neither closed under concatenation nor under Kleene-star.

Example 2.2. We extend the addition of $\mathbb{Z}$ to $M = \mathbb{Z} \cup \{e, a\}$ by setting $e + m = m$ for all $m \in M$, $a + a = 0$ and $a + k = k$ for all $k \in \mathbb{Z}$. Now, M forms a commutative monoid with neutral element e.

We claim that $L \in \mathrm{REC}(M)$ implies $L \cap \mathbb{Z} \in \mathrm{REC}(\mathbb{Z})$. Suppose $\varphi : M \to N$ is a homomorphism recognizing $L \subseteq M$, and let $\widetilde{\varphi} : \mathbb{Z} \to N$ be the restriction of φ to $\mathbb{Z}$. Then $F = \varphi(L)$ satisfies $\widetilde{\varphi}^{-1}(F) = \varphi^{-1}(F) \cap \mathbb{Z} = L \cap \mathbb{Z}$. This proves the claim.

Next we show $\{a\} \in \mathrm{REC}(M)$. Let $N = \{e, a, z\}$ be a commutative monoid with neutral element e with zero element z and with $a + a = z$. Let $\varphi : M \to N$ be the homomorphism defined by $e \mapsto e$, $a \mapsto a$, and $k \mapsto z$ for all $k \in \mathbb{Z}$. It satisfies $\varphi^{-1}(a) = \{a\}$. The product of $\{a\}$ with itself is $\{a\} + \{a\} = \{0\}$ and its Kleene-star is $\{a\}^* = \{e, a, 0\}$. Neither $\{0\}$ nor $\{e, a, 0\}$ is in $\mathrm{REC}(M)$, since in both cases the intersection with $\mathbb{Z}$ yields $\{0\}$ which by Proposition 2.2 is not in $\mathrm{REC}(\mathbb{Z})$. ◇

2.2. Syntactic monoids

The syntactic monoid $\mathrm{Synt}(L)$ of a subset $L \subseteq M$ is the unique minimal monoid which recognizes L. It is naturally equipped with a homomorphism $\varphi_L : M \to \mathrm{Synt}(L)$ which recognizes L, the so-called *syntactic homomorphism.* In order to formally define the syntactic monoid of L, we first introduce the *syntactic congruence* $\equiv_L$ over M. We set $u \equiv_L v$ if, for all $x, y \in M$, the following equivalence holds:

$$xuy \in L \Leftrightarrow xvy \in L.$$

One can think of (x, y) as a context, and then $u \equiv_L v$ means that u and v have the same behaviour with respect to all contexts. In particular, $\equiv_L$ forms an equivalence relation and the equivalence class of u is denoted by $[u]$. Suppose $u \equiv_L u'$ and $v \equiv_L v'$. For all $x, y \in M$ we have

$$xuvy \in L \Leftrightarrow xu'vy \in L \Leftrightarrow xu'v'y \in L,$$

where the first equivalence uses the context (x, vy) and the second equivalence uses the context (xu', y). This shows $uv \equiv_L u'v'$ and thus $\equiv_L$ is a congruence. The quotient $M/{\equiv_L} = \{\, [u] \mid u \in M \,\}$ is called the *syntactic monoid* of L, denoted by $\mathrm{Synt}(L)$. It is the set of all equivalence classes and the operation on $\mathrm{Synt}(L)$ is defined by $[u][v] = [uv]$. This is well-defined since $\equiv_L$ is a congruence. The natural projection $\varphi_L : M \to \mathrm{Synt}(L)$ with $\varphi_L(u) = [u]$ forms a homomorphism, the *syntactic homomorphism* of L. It recognizes L since

$$[u] \in \varphi_L(L) \Leftrightarrow u \equiv v \text{ for some } v \in L \Leftrightarrow u \in L.$$

The second equivalence uses the context $(1, 1)$. The following theorem shows that $\mathrm{Synt}(L)$ indeed is the minimal monoid recognizing L.

Theorem 2.4. *Let M and N be monoids, and let $\varphi : M \to N$ be a homomorphism recognizing $L \subseteq M$. Then $\varphi(u) \mapsto [u]$ defines a surjective homomorphism from the submonoid $\varphi(M)$ of N onto* $\mathrm{Synt}(L)$*. In particular, the following diagram commutes:*

$$\begin{tikzcd}
M \arrow[r, two heads, "\varphi"] \arrow[dr, two heads, "\varphi_L"'] & \varphi(M) \arrow[d, two heads, "\varphi(u) \mapsto [u]"] \\
 & \mathrm{Synt}(L)
\end{tikzcd}$$

Proof. We first show that $\varphi(u) \mapsto [u]$ is well-defined. Suppose $\varphi(u) = \varphi(v)$. For all $x, y \in M$ we have

$$xuy \in L \Leftrightarrow \varphi(xuy) \in L \Leftrightarrow \varphi(xvy) \in L \Leftrightarrow xvy \in L.$$

Thus $[u] = [v]$ as desired. Trivially, $\varphi(M) \to \mathrm{Synt}(L)$, $\varphi(u) \mapsto [u]$ is surjective, and it forms a homomorphism since $1 = \varphi(1) \mapsto [1]$ and $\varphi(u)\varphi(v) = \varphi(uv) \mapsto [uv] = [u][v]$. □

The above theorem says that $\mathrm{Synt}(L)$ is the homomorphic image of a submonoid of every monoid which recognizes L. If N is a finite monoid, then $|\mathrm{Synt}(L)| \leq |N|$. Therefore, $\mathrm{Synt}(L)$ is the unique minimal monoid which recognizes L.

Suppose $\varphi : M \to N$ is a surjective homomorphism which recognizes $L \subseteq M$. Let $P = \varphi(L)$. The syntactic congruence of P satsifies

$$\begin{aligned} u \equiv_L v &\Leftrightarrow \big(\forall x, y \in M \colon xuy \in L \Leftrightarrow xvy \in L\big) \\ &\Leftrightarrow \big(\forall x, y \in M \colon \varphi(xuy) \in \varphi(L) \Leftrightarrow \varphi(xvy) \in \varphi(L)\big) \\ &\Leftrightarrow \big(\forall x', y' \in N \colon x'\varphi(u)y' \in P \Leftrightarrow x'\varphi(v)y' \in P\big) \\ &\Leftrightarrow \varphi(u) \equiv_P \varphi(v). \end{aligned}$$

Here, the second equivalence holds since φ recognizes L, and the third equivalence holds since φ is surjective and since $P = \varphi(L)$. This shows that the syntactic monoids $\mathrm{Synt}(L)$ and $\mathrm{Synt}(P)$ are identical. Note that $\mathrm{Synt}(L)$ is a quotient of M and $\mathrm{Synt}(P)$ is a quotient of N. In particular, the homomorphism $N \to \mathrm{Synt}(L)$, $\varphi(u) \mapsto [u]$ given by Theorem 2.4 is the syntactic homomorphism of P. As a consequence, there exists a monoid M and a subset $L \subseteq M$ such that $N = \mathrm{Synt}(L)$ if and only if there exists $P \subseteq N$ such that N is the syntactic monoid of P.

2.3. Deterministic automata

There is a tight connection between homomorphisms and deterministic automata. In contrast to nondeterministic M-automata, the transition relation of a deterministic automaton defines a function $Q \times M \to Q$; and the easiest way for introducing deterministic M-automata (or just M-automata for short) is to rely on this function. The main technical advantage of this approach is that we do not have to deal with runs of the automaton. An *M-automaton* $\mathcal{A} = (Q, \cdot, q_0, F)$ consists of a set of *states* Q, of an *initial state* $q_0 \in Q$, of a set of *final states* $F \subseteq Q$, and of a *transition function* $\cdot : Q \times M \to Q$ satisfying

$$\begin{aligned} q \cdot 1 &= q \\ (q \cdot u) \cdot v &= q \cdot (uv) \end{aligned}$$

for all $q \in Q$ and all $u, v \in M$. The subset *accepted* by $\mathcal{A}$ is $L(\mathcal{A}) = \{\, u \in M \mid q_0 \cdot u \in F \,\}$. Remember that the subset accepted by a nondeterministic automaton is defined slightly more technical since it relies on the notion of a run. Suppose $\Sigma \subseteq M$ is a generating set. Then we can define the transition relation

$$\delta = \{\, (p, a, p \cdot a) \in Q \times \Sigma \times Q \mid p \in Q, a \in \Sigma \,\}.$$

The transition function $\cdot$ can be retrieved from δ since $q \cdot u$ is the unique state p such that there exists a run $qa_1q_1 \cdots a_{n-1}q_{n-1}a_np$ in the (a priori) nondeterministic M-automaton $(Q, \delta, \{q_0\}, F)$ with $u = a_1 \cdots a_n$, $a_i \in \Sigma$. In particular, M-automata can be graphically represented in the same way as nondeterministic M-automata. An M-automaton $\mathcal{A} = (Q, \cdot, q_0, F)$ is *finite* if it can be represented by a finite transition relation δ. If Q is finite, we say that the set of states of $\mathcal{A}$ is finite. If M is finitely generated and the set of states of $\mathcal{A}$ is finite, then $\mathcal{A}$ is finite. Conversely, if $\mathcal{A}$ is finite, then

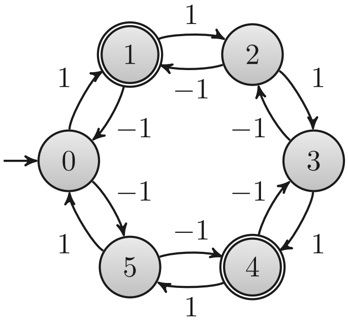
$$\{\, a \in M \mid (p, a, q) \in \delta \text{ for some } p, q \in Q \,\}$$

is a finite generating set of M and, moreover, we can replace the set of states Q by the finite set

$$\{q_0\} \cup \{\, p, q \in Q \mid (p, a, q) \in \delta \text{ for some } a \in M \,\}.$$

Frequently, we assume that all states are reachable, *i.e.*, for all $q \in Q$ there exists $u \in M$ with $q_0 \cdot u = q$ (otherwise we can replace Q by the reachable states).

Example 2.3. The set $\{1, -1,\}$ generates the integers $\mathbb{Z}$ with addition as operation. The $\mathbb{Z}$-automaton $\mathcal{A} = (\{\,0,1,2,3,4,5\,\}, \cdot, 0, \{\,1,4\,\})$ with $q \cdot k = (q+k) \bmod 6$ has the following graphical presentation:



It accepts the set $L(\mathcal{A}) = \{\, k + 6\ell \mid k \in \{1,4\}, \ell \in \mathbb{Z} \,\}$. ◇

2.4. Minimal automata

The minimal automaton of a subset $L \subseteq M$ is the unique smallest automaton accepting L. It can be seen a one-sided version of the syntactic monoid. Let $L(u) = u^{-1}L = \{\, y \in M \mid uy \in L \,\}$. Note that $L(1) = L$. We have $L(u) = L(v)$ if and only if for all $y \in M$ the following equivalence holds:

$$uy \in L \Leftrightarrow vy \in L.$$

The *minimal automaton* $\mathcal{A}_L = (Q_L, \cdot, q_{0L}, F_L)$ of L has states $Q_L = \{\, L(u) \mid u \in M \,\}$, its initial state is $q_{0L} = L(1) = L$, its set of final states is $F_L = \{\, L(u) \mid 1 \in L(u) \,\}$, and the transition function is defined by $L(u) \cdot v = L(uv)$. Suppose $L(u) = L(u')$. Then for all $y \in M$ we have

$$y \in L(uv) \Leftrightarrow uvy \in L \Leftrightarrow vy \in L(u) = L(u') \Leftrightarrow u'vy \in L \Leftrightarrow y \in L(u'v).$$

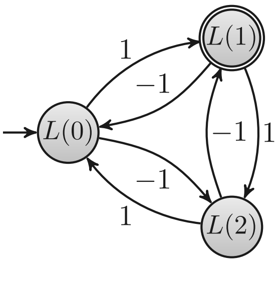
Thus $L(uv) = L(u'v)$ and the transition function of $\mathcal{A}_L$ is well-defined. Note that $L(u) \cdot 1 = L(u)$ and $(L(u) \cdot v) \cdot w = L(uvw) = L(u) \cdot (vw)$, *i.e.*, the mapping $\cdot$ indeed defines a transition function. The minimal automaton $\mathcal{A}_L$ accepts the set $L(\mathcal{A}_L) = L$ since $L(1) \cdot u = L(u)$ and

$$u \in L \Leftrightarrow 1 \in L(u).$$

Example 2.4. We consider the subset $L = \{\, k + 6\ell \mid k \in \{\,1,4\,\}, \ell \in \mathbb{Z} \,\}$ from Example 2.3. Adding or subtracting multiples of 3 does not change membership in L. It therefore suffices to determine $L(0)$, $L(1)$, and $L(2)$:

$$\begin{aligned} L(0) &= 1 + 3\mathbb{Z} = \{\, 1 + 3k \mid k \in \mathbb{Z} \,\} \\ L(1) &= 3\mathbb{Z} \\ L(2) &= -1 + 3\mathbb{Z} \end{aligned}$$

Thus, the minimal automaton $\mathcal{A}_L$ of L is given by



The initial state is $L(0)$ and not $L(1)$ since 0 is the neutral element of $(\mathbb{Z},+)$. Also note that, for example, the transition $L(0)\cdot 1$ yields the set $L(0)-1=L(1)$ since $L(0)-1$ is the residual of $L(0)$ by 1. ◇

The following theorem shows that the minimal automaton of L is the unique minimal automaton which accepts L.

Theorem 2.5. *Let M be a monoid and let $\mathcal{A}=(Q,\cdot,q_0,F)$ be an M-automaton with $L(\mathcal{A})=L$. Then $q_0\cdot u\mapsto L(u)$ defines a surjective mapping from the reachable states of $\mathcal{A}$ onto the states Q_L of the minimal automaton.*

Proof. If $p=q_0\cdot u$, then $L(u)=\{\,v\in M\mid p\cdot v\in F\,\}$. Since $\{\,v\in M\mid p\cdot v\in F\,\}$ does not depend on u, the mapping $q_0\cdot u\mapsto L(u)$ is well-defined. Some preimage of $L(u)$ is the state $q_0\cdot u$. □

For an M-automaton $\mathcal{A}=(Q,\cdot,q_0,F)$ and a state $p\in Q$ one can define the set $L(p)=\{\,v\in M\mid p\cdot v\in F\,\}$. It is the subset of M accepted by the M-automaton $\mathcal{A}_p=(Q,\cdot,p,F)$ with initial state p. The proof of Theorem 2.5 shows that for every reachable state p of $\mathcal{A}$, the set $L(p)$ is a state of the minimal automaton. The *Myhill-Nerode equivalence relation* $\equiv_{\mathcal{A}}$ on Q is defined by $p\equiv_{\mathcal{A}} q$ if $L(p)=L(q)$. The minimal automaton for $L(\mathcal{A})$ can be obtained by identifying Myhill-Nerode equivalent states. This fact is often used by minimization algorithms. Moreover, an M-automaton $\mathcal{A}=(Q,\cdot,q_0,F)$ is minimal if and only if the following two properties hold:

- All states in Q are reachable, *i.e.*, we have $q_0\cdot M=Q$.
- For any two distinct states states $p,q\in Q$ we have $L(p)\neq L(q)$.

2.5. Transition monoids and Cayley automata

In this section we show that every M-automaton $\mathcal{A}$ can be transformed into a homomorphism $\varphi_{\mathcal{A}}:M\to T_{\mathcal{A}}$ which recognizes $L(\mathcal{A})$. Conversely, for every homomorphism $\varphi:M\to N$ and every subset $P\subseteq N$ one can construct an M-automaton $\mathcal{A}_\varphi$ such that $L(\mathcal{A}_\varphi)=\varphi^{-1}(P)$. Moreover, both translations preserve finiteness in the sense that the automaton has finitely many states if and only if the monoid N (resp. $T_{\mathcal{A}}$) is finite.

Let $\mathcal{A}=(Q,\cdot,q_0,F)$ be an M-automaton. Every element $u\in M$ defines a mapping $\delta_u:Q\to Q$ by setting $\delta_u(q)=q\cdot u$. When defining the product of two such functions as $\delta_u\delta_v=\delta_{uv}=\delta_v\circ\delta_u$, then $T_{\mathcal{A}}=\{\,\delta_u\mid u\in M\,\}$ forms a monoid, the transition monoid of $\mathcal{A}$. It is a submonoid of the set of all functions $Q\to Q$. In particular, if Q is finite, then the transition monoid of $\mathcal{A}$ is finite. The mapping $\varphi_{\mathcal{A}}:M\to T_{\mathcal{A}}$ defined by $\varphi_{\mathcal{A}}(u)=\delta_u$ recognizes $L(\mathcal{A})$:

$$u\in L(\mathcal{A})\Leftrightarrow q_0\cdot u\in F\Leftrightarrow \delta_u(q_0)\in F\Leftrightarrow \delta_u\in\{\,\delta_v\mid\delta_v(q_0)\in F\,\}\,.$$

This shows that $L(\mathcal{A})$ is the inverse image of $\{\,\delta_v\mid\delta_v(q_0)\in F\,\}\subseteq T_{\mathcal{A}}$ under the homomorphism $\varphi_{\mathcal{A}}$. Note that both the initial state and the final states are encoded in the accepting set for $\varphi_{\mathcal{A}}$.

Theorem 2.6. *Let $L\subseteq M$, let $\mathcal{A}_L$ be its minimal M-automaton, and let T_L be the transition monoid of $\mathcal{A}_L$. Then $T_L\to\mathrm{Synt}(L)$, $\delta_u\mapsto[u]$ defines an isomorphism between T_L and the syntactic monoid* $\mathrm{Synt}(L)$.

Proof. By Theorem 2.4 the mapping $\delta_u\mapsto[u]$ is a surjective homomorphism. It remains to show injectivity. Suppose $u\equiv_L v$. For $x\in M$ we have $\delta_u(L(x))=L(x)\cdot u=L(xu)$. Thus for all $y\in M$ we have

$$y\in\delta_u(L(x))=L(xu)\Leftrightarrow xuy\in L\Leftrightarrow xvy\in L\Leftrightarrow y\in L(xv)=\delta_v(L(x))$$

and thus $\delta_u(L(x))=\delta_v(L(x))$. Since this holds for all $x\in M$ we conclude $\delta_u=\delta_v$. □

If $\mathcal{A}$ is a finite automaton with n states, then its transition monoid has at most n^n elements. The following example of Holzer and König shows that this bound is tight [**?**].

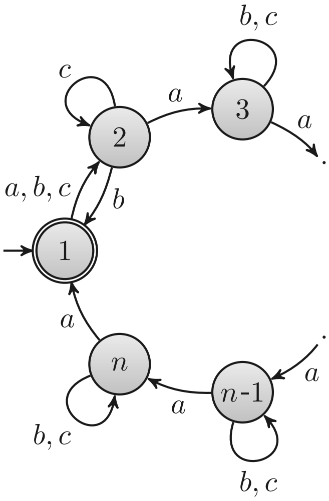
Example 2.5. We define an automaton with states $Q = \{1, \ldots, n\}$. We use the notation

$$\tau = \begin{pmatrix} 1 & 2 & \cdots & n \\ \tau(1) & \tau(2) & \cdots & \tau(n) \end{pmatrix}$$

to define a mapping $\tau : Q \to Q$. Such mappings are also called *transformations* on Q. Let

$$\alpha = \begin{pmatrix} 1 & 2 & \cdots & n-1 & n \\ 2 & 3 & \cdots & n & 1 \end{pmatrix}, \qquad \beta = \begin{pmatrix} 1 & 2 & 3 & \cdots & n \\ 2 & 1 & 3 & \cdots & n \end{pmatrix}, \qquad \gamma = \begin{pmatrix} 1 & 2 & 3 & \cdots & n \\ 2 & 2 & 3 & \cdots & n \end{pmatrix}.$$

Every permutation on Q is generated by the two permutations α and β. The mapping γ allows to identify two elements. Thus the set of all transformations on Q is generated by $\{\alpha, \beta, \gamma\}$. Let $\mathcal{A} = (Q, \cdot, 1, \{1\})$ be the $\{a, b, c\}^*$-automaton defined by $i \cdot a = \alpha(i)$, $i \cdot b = \beta(i)$, and $i \cdot c = \gamma(i)$ for $i \in Q$. Its graphic representations is



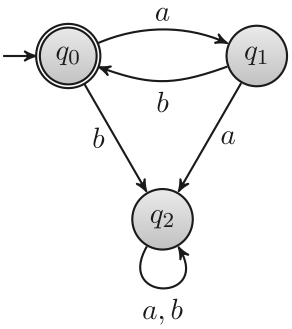
The automaton $\mathcal{A}$ is minimal since all states are reachable, and for states $i \neq j$ in Q we have $i \cdot a^{n-i+1} = 1$ and $j \cdot a^{n-i+1} \neq 1$. The transition monoid of $\mathcal{A}$ has n^n elements because there are exactly n^n transformations on Q. By Theorem 2.6, the transition monoid of $\mathcal{A}$ is the syntactic monoid of $L(\mathcal{A})$. In particular, by Theorem 2.4 there is no smaller monoid recognizing $L(\mathcal{A})$. $\diamond$

The step from homomorphisms to automata is straightforward. Suppose $\varphi : M \to N$ is a homomorphism and P is a subset of N. The *Cayley automaton* of φ and P is $\mathcal{A}_\varphi = (N, \cdot, 1, P)$ with transition function $n \cdot u = n\varphi(u)$ for $n \in N$ and $u \in M$. We have

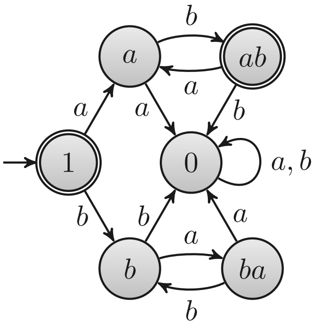
$$u \in L(\mathcal{A}_\varphi) \Leftrightarrow \varphi(u) = 1 \cdot u \in P,$$

i.e., the M-automaton $\mathcal{A}_\varphi$ accepts $\varphi^{-1}(P)$. In particular, for $L = \varphi^{-1}(P)$ it accepts the subset $L \subseteq M$. If $\varphi : M \to \mathrm{Synt}(L)$ is the syntactic homomorphism of $L \subseteq M$, then in general the Cayley automaton $\mathcal{A}_\varphi$ is not the minimal automaton of L. This means that if we first construct a homomorphism φ for some given automaton $\mathcal{A}$ and afterwards translate φ into an automaton $\mathcal{A}_\varphi$, then in general we have $\mathcal{A} \neq \mathcal{A}_\varphi$.

Example 2.6. Let $M = \{a, b\}^*$ and let $L = (ab)^*$. The minimal automaton of L is:



We have $L(q_0) = (ab)^*$, $L(q_1) = b(ab)^*$ and $L(q_2) = \{a, b\}^*$. The state q_2 is called a *sink state* since $q_2 \cdot u = q_2$ for all $u \in \{a, b\}^*$. The syntactic monoid of L can be represented by shortest words in their respective $\equiv_L$-classes. Using this notation we have $\mathrm{Synt}(L) = \{ 1, a, b, ab, ba, 0 \}$. As usual, 1 is the neutral element, and the element 0 satisfies $0x = x0 = 0$ for all $x \in \mathrm{Synt}(L)$. We have $0 = aa = bb$. The remaining entries of the multiplication table are given by the rules $aba = a$ and $bab = b$ since $aba \equiv_L a$ and $bab \equiv_L b$. For instance, we have $ab \cdot ba = a \cdot bb \cdot a = a \cdot 0 \cdot a = 0$ or $ab \cdot ab = aba \cdot b = a \cdot b = ab$. Suppose $\varphi : \{a, b\}^* \to \mathrm{Synt}(L)$ is the syntactic homomorphism of L, *i.e.*, we have $a \mapsto a$ and $b \mapsto b$. Then $L = \varphi^{-1}(\{1, ab\})$ and the Cayley automaton $\mathcal{A}_\varphi$ is:



In particular $\mathcal{A}_\varphi$ is not the minimal automaton of L. ◇

On the other hand, let $\varphi : M \to N$ be a surjective homomorphism and let $\mathcal{A}_\varphi$ be its Cayley automaton. Then the transition monoid of $\mathcal{A}_\varphi$ is N since the mapping $\delta_u \mapsto \varphi(u)$ is an isomorphism between the transtition monoid of $\mathcal{A}_\varphi$ and N. This can be seen as follows. It is well-defined because $\delta_u = \delta_v$ implies $\varphi(u) = \delta_u(1) = \delta_v(1) = \varphi(v)$; it injective because $\varphi(u) = \varphi(v)$ implies $\delta_u(n) = n \cdot u = n\varphi(u) = n\varphi(v) = n \cdot v = \delta_v(n)$ for all $n \in N$, *i.e.*, we have $\delta_u = \delta_v$. The mapping $\delta_u \mapsto \varphi(u)$ trivially defines a homomorphism and it is surjective because φ is surjective.

Remark 2.1. Let N be a monoid and let $\Sigma \subseteq N$ be a generating set. Then the *Cayley graph* of N is $G_N = (N, E)$ with Σ-labeled edges $\{ (n, a, n \cdot a) \mid n \in N,\ a \in \Sigma \}$. Sometimes G_N is called the *right* Cayley graph of N, whereas the edges $\{ (n, a, a \cdot n) \mid n \in N,\ a \in \Sigma \}$ define the *left Cayley graph.* If $\mathrm{id} : N \to N$ is the identity mapping, then (after forgetting about initial and final states) the Cayley automaton $\mathcal{A}_{\mathrm{id}}$ can be identified with the Cayley graph G_N. If we only draw Σ-transitions, then the graph structure of $\mathcal{A}_{\mathrm{id}}$ is G_N. ◇

2.6. The Myhill-Nerode Theorem

We are now ready to prove that recognition by finite monoids and acceptance by deterministic automata with finitely many states define the same subsets. Moreover, one can restate this property in terms of minimal automata and syntactic monoids. This following result is known as the Myhill-Nerode Theorem.

Theorem 2.7 (Myhill, Nerode)**.** *Let M be monoid and let $L \subseteq M$. The following conditions are equivalent:*

(a) *$L \in \mathrm{REC}(M)$, i.e., L is recognized by a finite monoid.*

(b) *L is accepted by an M-automaton with finitely many states.*

(c) *The minimal automaton of L has finitely many states.*

(d) *The syntactic monoid of L is finite.*

Proof. (a) ⇒ (b): If the homomorphism $\varphi : M \to N$ for some finite monoid N recognizes L, then the Cayley automaton $\mathcal{A}_\varphi$ has finitely many states and it accepts L. (b) ⇒ (c): By Theorem 2.5, the minimal automaton of L is the smallest automaton recognizing L. (c) ⇒ (d): By Theorem 2.6, the syntactic monoid is the transition monoid of the minimal automaton. (d) ⇒ (a): The syntactic monoid $\mathrm{Synt}(L)$ recognizes L. □

As a corollary we obtain the following characterization of subsets accepted by finite M-automata.

Corollary 2.8. *Let M be monoid and let $L \subseteq M$. The following conditions are equivalent:*

(a) *M is finitely generated and $L \in \mathrm{REC}(M)$.*

(b) *L is accepted by a finite M-automaton.*

Proof. (a) ⇒ (b): Suppose M is generated by a finite set Σ. By the Myhill-Nerode Theorem, the set L is accepted by an M-automaton $\mathcal{A} = (Q, \cdot, q_0, F)$ with finitely many states. By representing the transition function $\cdot$ by $\delta = \{\, (p, a, p \cdot a) \mid p \in Q,\ a \in \Sigma \,\}$ we see that $\mathcal{A}$ is finite.

(b) ⇒ (a): If L is accepted by a finite M-automaton $\mathcal{A} = (Q, \delta, q_0, F)$, then by replacing Q by the reachable states $q_0 \cdot M$ we obtain an automaton for L with finitely many states. By the Myhill-Nerode Theorem, we have $L \in \mathrm{REC}(M)$. Moreover, the finite set

$$\Sigma = \{\, a \in M \mid (p, a, q) \in \delta \text{ for some } p, q \in Q \,\}$$

generates M (since, for instance, δ defines the transition function $q_0 \cdot u$ for all $u \in M$). □

Figure 2.1 summarizes the transformations between automata and monoids; the terms n and n^n indicate the maximal blow-up relative to the number of states and the size of the monoid.

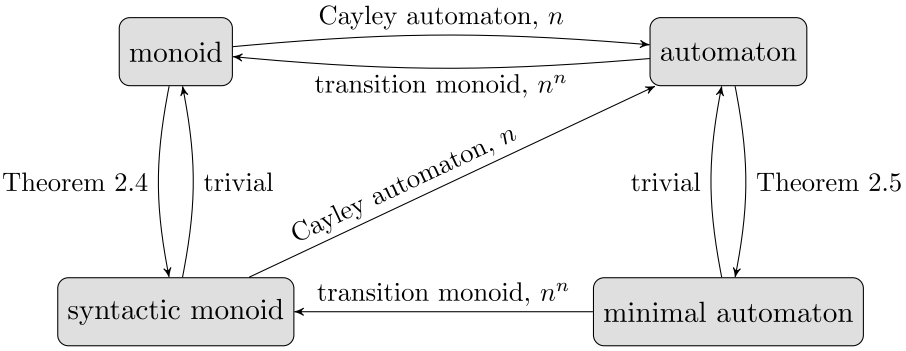


Figure 2.1.: Transformation between automata and monoids.

2.7. Learning Recognizable Sets

We present Angluin's L^*-learning algorithm [**?**]. The setting is as follows. Let M be generated by Σ. A *teacher* knows a recognizable set $L \subseteq M$ and a *learner* wants to learn the minimal automaton of L. The learner knows M and Σ and he can ask two kinds of questions to the teacher:

Membership queries: For $u \in M$ the learner can ask whether $u \in L$. The answer of the teacher is either yes or no.

Equivalence queries: For an M-automaton $\mathcal{A}$ the learner can ask whether $L(\mathcal{A}) = L$. The answer of the teacher is either yes or he provides a element $u = a_1 \cdots a_k$ with $a_i \in \Sigma$ in the symmetric difference of $L(\mathcal{A})$ and L.

Let n be the number of states of the minimal automaton of L. A naive approach for the learner could be enumerating all M-automata and using each of them for an equivalence query; eventually, the teacher will say yes. This approach is exponential in n due to the large number of automata with at most n states. Angluin's L^*-algorithm yields an algorithm using a polynomial number of queries (at least if the teacher provides counter-examples of small length in his answers to equivalence queries). For elements $u, v \in M$ we define $u \approx_L v$ if for all $y \in M$ we have

$$uy \in L \Leftrightarrow vy \in L.$$

The relation $\approx_L$ defines an equivalence relation on M and its classes are the states of the minimal automaton. The learner tries to approximate $\approx_L$. A set of *extensions* is a subset $E \subseteq M$ with $1 \in E$. For a set of extensions E we set $u \sim_E v$ if for all $e \in E$ we have

$$ue \in L \Leftrightarrow ve \in L.$$

Note that $u \approx_L v$ implies $u \sim_E v$ for all sets of extensions E. If $u \sim_E v$, then

$$u \in L \Leftrightarrow v \in L$$

since $1 \in E$. A subset $S \subseteq M$ is a set of *samples* if $1 \in S$ and S prefix-closed. A subset $S \subseteq M$ is *prefix-closed* if every element $u \in S$ can be written as $u = a_1 \cdots a_k$ for $a_i \in \Sigma$ such that $\{\, a_1 \cdots a_i \mid 0 \leq i \leq k \,\} \subseteq S$. A pair $P = (S, E)$ consisting of samples S and extensions E is *complete* if for all elements $s \in S$ and for all generators $a \in \Sigma$ there exists $t \in S$ with

$$sa \sim_E t.$$

The pair P is *consistent* if for all elements $s, t \in S$ and for all generators $a \in \Sigma$ we have

$$s \sim_E t \Rightarrow sa \sim_E ta.$$

For a set of extensions E and an element $u \in M$ we define $[u]_E = \{\, v \in M \mid u \sim_E v \,\}$. If $P = (S, E)$ is both complete and consistent, then we obtain an automaton $\mathcal{A}_P = (Q_P, \cdot, q_P, F_P)$ with

$$\begin{aligned} Q_P &= \{\, [s]_E \mid s \in S \,\}, \\ [s]_E \cdot a &= [sa]_E \quad \text{for } s \in S, \\ q_P &= [1]_E, \\ F_P &= \{\, [s]_E \mid s \in S \cap L \,\}. \end{aligned}$$

If $s \in S$ and $a \in \Sigma$, then $[sa]_E \in Q_P$ since P is complete. If $[s]_E = [t]_E$ for $s, t \in S$, then by consistency of P we obtain $[sa]_E = [ta]_E$. This shows that the transition function is well-defined.

Lemma 2.9. *If $P = (S, E)$ is both complete and consistent, then $L(\mathcal{A}_P) \cap S = L \cap S$.*

Proof. Let $u \in S$. Since S is prefix-closed we can write $u = a_1 \cdots a_k$ with $a_i \in \Sigma$ such that $a_1 \cdots a_i \in S$ for all $0 \leq i \leq k$. Therefore, there exists a run

$$[1]_E \xrightarrow{a_1} [a_1]_E \xrightarrow{a_2} [a_1 a_2]_E \xrightarrow{a_3} \cdots \xrightarrow{a_{k-1}} [a_1 \cdots a_{k-1}]_E \xrightarrow{a_k} [a_1 \cdots a_k]_E$$

in $\mathcal{A}_P$. This run is accepting if and only $u \in L$. □

We can consider the states $Q_P = \{ [u]_E \mid u \in S \}$ also for pairs $P = (S, E)$ which are not complete or not consistent. If $P = (S, E)$ and $P' = (S', E)$ for $S \subseteq S'$, then $Q_P \subseteq Q_{P'}$. If $P = (S, E)$ and $P' = (S, E')$ for $E \subseteq E'$, then for every $[s]_{E'} \in Q_{P'}$ with $s \in S$ we have $[s]_E \in Q_P$ and $[s]_{E'} \subseteq [s]_E$, *i.e.*, $Q_{P'}$ refines Q_P. This shows that by increasing S or E, the number of states never decreases.

Lemma 2.10. *Let $P = (S, E)$ be complete and consistent, and let $u = a_1 \cdots a_k$ with $a_i \in \Sigma$ be an element in the symmetric difference of $L(\mathcal{A}_P)$ and L. Then $P' = (S', E)$ with $S' = S \cup \{ a_1 \cdots a_i \mid 1 \leq i \leq k \}$ is not consistent.*

Proof. Suppose P' were consistent. For all $i \in \{1, \ldots, k\}$ there exists $s_i \in S$ with

$$[1]_E \cdot (a_1 \cdots a_i) = [s_i]_E.$$

In particular, we have $s_i a_i \sim_E s_{i+1}$. By induction on i, consistency of P' yields $s_i \sim_E a_1 \cdots a_i$. In particular $s_k \sim_E u$. This leads to the following contradiction:

$$\begin{aligned} u \in L(\mathcal{A}_P) &\Leftrightarrow s_k \in L(\mathcal{A}_P) && \text{since } [1]_E \cdot u = [1]_E \cdot s_k \\ &\Leftrightarrow s_k \in L && \text{by Lemma 2.9} \\ &\Leftrightarrow u \in L && \text{by } s_k \sim_E u \text{ and } 1 \in E \\ &\Leftrightarrow u \notin L(\mathcal{A}_P) && \text{by choice of } u. \end{aligned}$$

Therefore, P' is not consistent. □

The algorithm

Angluin's learning algorithm now proceeds as follows. We start with some arbitrary pair $P = (S, E)$ of samples and extensions; for instance we could use $S = E = \{1\}$. Then we successively replace P by the pair P' defined by the following procedure:

The pair P is not complete: Let $s \in S$ and $a \in \Sigma$ with $[sa]_E \notin Q_P$. Then we set $P' = (S', E)$ with $S' = S \cup \{sa\}$. This yields $|Q_{P'}| > |Q_P|$ since $[sa]_E \in Q_{P'}$.

The pair P is not consistent: Let $s, t \in S$ and $a \in \Sigma$ with $s \sim_E t$ and $sa \not\sim_E ta$. There exists $e \in E$ with $sae \in L \Leftrightarrow tae \notin L$. Then we set $P' = (S, E')$ with $E' = E \cup \{ae\}$. This yields $|Q_{P'}| > |Q_P|$ since $[s]_{E'} \neq [t]_{E'}$.

The pair P is complete and consistent, but $L(\mathcal{A}_P) \neq L$: Let $u = a_1 \cdots a_k$ with $a_i \in \Sigma$ be an element in the symmetric difference of $L(\mathcal{A}_P)$ and L. Then we set $P' = (S', E)$ with $S' = S \cup \{ a_1 \cdots a_i \mid 1 \leq i \leq k \}$. By Lemma 2.10, the pair P' is not consistent. Therefore, the set of states increases in the next iteration.

Note that we preserve the invariant that the set of samples S is prefix-closed. It is sufficient for the learner to represent the restriction of the relation $\sim_E$ to the set $S \cup S\Sigma$.

The number of queries

The learner asks membership queries for all elements se with $s \in S \cup S\Sigma$ and $e \in E$. This suffices for checking whether the current pair P is complete and consistent and for computing the automaton $\mathcal{A}_P$. In the case that the current pair is both complete and consistent, he asks an equivalence query.

We assume that the learner starts with the pair $(\{1\}, \{1\})$. Throughout the number of states $|Q_P|$ is bounded by n. Since increasing the extensions also increases the number of states, we have $|E| \leq n$. If all counter-examples $u = a_1 \cdots a_k$ of the teacher satisfy $k \leq m$, then after adding at most $m + 1$ elements to S, the number of states increases. Thus $|S| \leq (m + 1)n$. This shows that the learner asks at most

$$\big((m + 1)n + |\Sigma| \, (m + 1)n\big)n \;\in\; \mathcal{O}(|\Sigma| \, mn^2)$$

membership queries. If the teacher provides counter-examples of minimal length, then by Corollary 4.6 we will see that $m \leq |Q_P| + n - 2 \leq 2n - 2$. In particular, the algorithm requires at most $\mathcal{O}(|\Sigma|\, n^3)$ membership queries. The number of equivalence queries is at most n since every equivalence query causes an increase in the number of states.

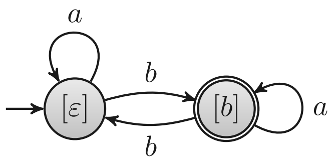
Example 2.7. We apply Angluin's algorithm for learning the language $L = b\,\{ab, a\}^* \subseteq \Sigma^*$ for $\Sigma = \{a, b\}$, see Example 1.7. We represent the restriction of the relation $\sim_E$ to the set $S \cup S\Sigma$ as a table. The top rows correspond to S, the bottom rows correspond to $S\Sigma \setminus S$, and the columns correspond to E. At coordinate (s, e) for $s \in S \cup S\Sigma$ and $e \in E$ we write 1 if $se \in E$ and 0 otherwise. Two elements in $S \cup S\Sigma$ are $\sim_E$-equivalent if and only if they have identical 0-1 rows. The initial table for the pair $P = (\{\varepsilon\}, \{\varepsilon\})$ is as follows.

	ε
ε	0
a	0
b	1

The pair P is not complete since the 0-1-row for b in the lower half does not occur in the upper half. We thus add b to S which yields the following situation.

	ε
ε	0
b	1
a	0
ba	1
bb	0

Now, the pair is complete and consistent and the learner asks an equivalence query with the following automaton.



Since the automaton is not correct, the teacher returns the counter-example ab. Incorporating ab and its prefix a into S yields the following table.

	ε
ε	0
a	0
b	1
ab	0
aa	0
ba	1
bb	0
aba	0
abb	0

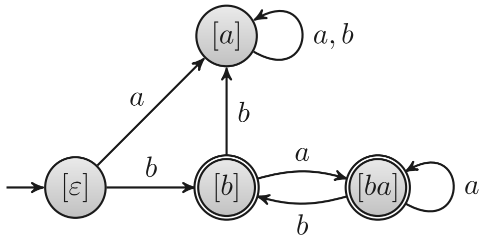
The current pair P is not consistent since ε and a are $\sim_E$-equivalent (they have the same 0-1-row) but $\varepsilon \cdot b \not\sim_E a \cdot b$. Therefore, we add b to E.

	ε	b
ε	0	1
a	0	0
b	1	0
ab	0	0
aa	0	0
ba	1	1
bb	0	0
aba	0	0
abb	0	0

This table is not complete since the 0-1-row of ba does not occur in the upper half. We therefore add ba to S.

	ε	b
ε	0	1
a	0	0
b	1	0
ab	0	0
ba	1	1
aa	0	0
bb	0	0
aba	0	0
abb	0	0
baa	1	1
bab	1	0

This table is both complete and consistent. Therefore, the learner uses the following automaton for an equivalence query.



The teacher answers *yes* since this automaton accepts L. ◇

Learning Boolean functions with few terms

Let $\mathbb{B} = \{1, 0\}$ be the Boolean lattice. As usual, 1 is for *true* and 0 is for *false*. A *Boolean function* is a mapping of the form $f : \mathbb{B}^n \to \mathbb{B}$ for some $n \in \mathbb{N}$. Boolean formulas are one way of defining Boolean functions. We assume to have a set of *variables* $x_1, \ldots, x_n$. A *literal* is a variable or the negation of a variable. A *term* is a conjunction of literals, but it never contains both a variable and its negation. We consider formulas φ which are Boolean combinations of terms $t_1, \ldots, t_k$. For $b_1, \ldots, b_n \in \mathbb{B}$ we let $\varphi(b_1, \ldots, b_n)$ be the value obtained when interpreting the variable x_i by b_i. This way, every formula φ defines a Boolean function $f_\varphi : b_1 \cdots b_n \mapsto \varphi(b_1, \ldots, b_n)$. Suppose the teacher knows a Boolean function f_φ for some Boolean combination φ of terms $t_1, \ldots, t_k$ and the learner wants to learn the function f_φ. The learner can ask two kinds of queries.

- He can ask the teacher for the value $f_\varphi(b_1, \ldots, b_n)$ for $b_i \in \mathbb{B}$.

- Or he can ask whether $g = f_\varphi$ for some Boolean function g. We do not care about the presentation of g at this point. If $g \neq f_\varphi$, then the teacher provides $b_1 \cdots b_n \in \mathbb{B}^n$ with $g(b_1, \ldots, b_n) \neq f_\varphi(b_1, \ldots, b_n)$.

We present Kushilevitz's reduction to Angluin's L^*-algorithm [**?**]. To this end, we represent $g : \mathbb{B}^n \to \mathbb{B}$ as the language

$$L_g = \{\, b_1 \cdots b_n \in \mathbb{B}^n \mid g(b_1, \ldots, b_n) = 1 \,\} \subseteq \mathbb{B}^*.$$

If $g = f_\varphi$, then we write L_φ instead of L_{f_φ}. Now, the learner wants to learn the minimal automaton of L_φ. This uniquely defines f_φ. Membership queries and equivalence queries immediately translate into queries of the above type. Note that the answer to membership queries for $u \in \mathbb{B}^* \setminus \mathbb{B}^n$ is always *no* and that the learner can provide counter-examples for himself if the current language is not contained in $\mathbb{B}^n$.

Next, we give an automaton $\mathcal{A}_\varphi = (Q, \cdot, q_0, F)$ for the language L_φ. Let Q be the disjoint union $\{0, \ldots, n\} \times 2^{\{1,\ldots,k\}}$ and $\{q_{\text{sink}}\}$. The semantics of a state $(i, R) \in Q$ is that the automaton has read i bits and that the terms in R are not yet false. The state q_{sink} for processing more than n bits. The initial state is $q_0 = (0, \{1, \ldots, k\})$ and the final states F are those pairs (n, R) such φ is true when setting $t_i = 1$ if $i \in R$. The transition function is defined by

$$(i-1, R) \cdot a = \begin{cases} \big(i, R \setminus \{\, j \mid t_j \text{ contains the literal } x_i \,\}\big) & \text{if } a = 0 \\ \big(i, R \setminus \{\, j \mid t_j \text{ contains the negation of } x_i \,\}\big) & \text{if } a = 1 \end{cases}$$

and $q_{\text{sink}} \cdot a = q_{\text{sink}}$ for all $a \in \mathbb{B}$. The automaton $\mathcal{A}_\varphi$ accepts L_φ and it has $(n+1)2^k+1$ states. By Angluin's L^*-algorithm, the number queries is polynomial in n and 2^k. In particular, if $k \in \mathcal{O}(\log n)$, then the resulting algorithm requires only polynomially many queries.

3. Regular Languages

Kleene's Theorem states that, for free monoids over finite alphabets, the rational and the recognizable languages coincide. It is therefore common to refer to this language class as the *regular languages.* Rational sets have some closure properties and recognizable sets have other closure properties, see Table 3.1. An important consequence of Kleene's Theorem is that regular languages have both the closure properties of rationals sets and the closure properties of recognizable sets.

3.1. McKnight's Theorem

Essentially, McKnight's Theorem gives the first half of Kleene's Theorem. Every finite deterministic M-automaton is also a finite nondeterministic M-automaton. This does not immediately show that all recognizable sets are rational since the automaton model for recognizability relies on a slightly different notion of finiteness, namely on a finite number of states. On the other hand, finite nondeterministic automata have a finite number of transitions. We have already seen that for finitely generated monoids these two aspects of finiteness coincide. This observations yields the following theorem.

Theorem 3.1 (McKnight)**.** *Let M be a monoid. Then M is finitely generated if and only if* $\mathrm{REC}(M) \subseteq \mathrm{RAT}(M)$.

Proof. Suppose M is finitely generated. If $L \in \mathrm{REC}(M)$, then by Corollary 2.8 the set L is accepted by a finite M-automaton. Theorem 1.7 then yields $L \in \mathrm{RAT}(M)$.

For the other inclusion let $\mathrm{REC}(M) \subseteq \mathrm{RAT}(M)$. Since $M \in \mathrm{REC}(M)$, this yields $M \in \mathrm{RAT}(M)$. Now, by Proposition 1.1 the monoid M is finitely generated. □

3.2. The powerset construction and Kleene's Theorem

We will give two different proofs of Kleene's Theorem. The first proof relies on the so-called powerset construction for nondeterministic automata whereas the second proof uses an implementation of nondeterministic automata in terms of Boolean matrices. This second proof will be given in the next section.

Here, we show that for every nondeterministic word automaton there exists an equivalent deterministic automaton, the so-called powerset construction. Let Σ be an alphabet and let $\mathcal{A} = (Q, \delta, I, F)$ be a nondeterministic Σ^*-automaton with $\delta \subseteq Q \times \Sigma \times Q$. We introduce a transition function on the powerset 2^Q of Q as follows. For $P \subseteq Q$ and $a \in \Sigma$ we define

$$P \cdot a = \{\, q \in Q \mid \exists p \in P \colon (p, a, q) \in \delta \,\}\,.$$

That is, $P \cdot a$ consists of the states which are reachable from P using an a-transition. The transition function is extended to words by defining $P \cdot au = (P \cdot a) \cdot u$ for $a \in \Sigma$ and $u \in \Sigma^*$. The *powerset construction* of $\mathcal{A}$ is $\mathcal{P}(\mathcal{A}) = (Q', \cdot, I, F')$ with states

$$Q' = \{\, P \subseteq Q \mid \exists u \in \Sigma^* \colon I \cdot u = P \,\}$$

and final states $F' = \{\, P \in Q' \mid P \cap F \neq \emptyset \,\}$. This means that the states of $\mathcal{P}(\mathcal{A})$ are the reachable subsets of Q and the final states are those which contain some final state of $\mathcal{A}$.

Proposition 3.2. *Let Σ be an alphabet and let $\mathcal{A} = (Q, \delta, I, F)$ be a nondeterministic Σ^*-automaton with $\delta \subseteq Q \times \Sigma \times Q$. Then the powerset construction $\mathcal{P}(\mathcal{A})$ is a deterministic Σ^*-automaton with $L(\mathcal{P}(\mathcal{A})) = L(\mathcal{A})$.*

Proof. Let $\mathcal{P}(\mathcal{A}) = (Q', \cdot, I, F')$. Since the first letter in every nonempty word is unique, the transition function $\cdot : Q' \times \Sigma^* \to Q'$ is well-defined. Moreover, $P \cdot \varepsilon = P$ and $P \cdot uv = (P \cdot u) \cdot v$. Thus $\mathcal{P}(\mathcal{A})$ is a Σ^*-automaton. By induction on the length of u we show

$$I \cdot u = \left\{ q \in Q \;\middle|\; \exists q_0 \in I \colon q_0 \xrightarrow{u} q \right\}.$$

For $u = \varepsilon$, the claim is true. Let now $u = u'a$ for $u' \in \Sigma^*$ and $a \in \Sigma$. By induction, $I \cdot u'$ has the desired form. This yields

$$\begin{aligned} I \cdot u = (I \cdot u') \cdot a &= \left\{ q' \in Q \;\middle|\; \exists q_0 \in I \colon q_0 \xrightarrow{u'} q' \right\} \cdot a \\ &= \left\{ q \in Q \;\middle|\; \exists q_0 \in I \, \exists q' \in Q \colon q_0 \xrightarrow{u'} q' \text{ and } (q', a, q) \in \delta \right\} \\ &= \left\{ q \in Q \;\middle|\; \exists q_0 \in I \colon q_0 \xrightarrow{u} q \right\}. \end{aligned}$$

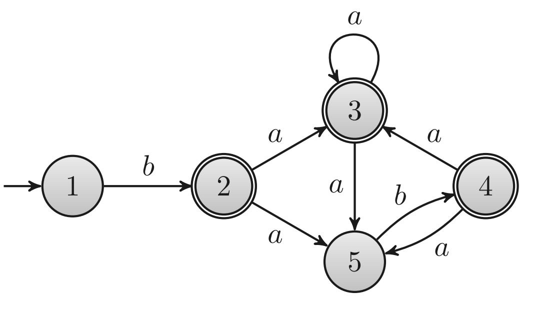
Therefore $I \cdot u \in F'$ if and only if $\mathcal{A}$ has an accepting run on u. This shows that $\mathcal{P}(\mathcal{A})$ and $\mathcal{A}$ accept the same language. □

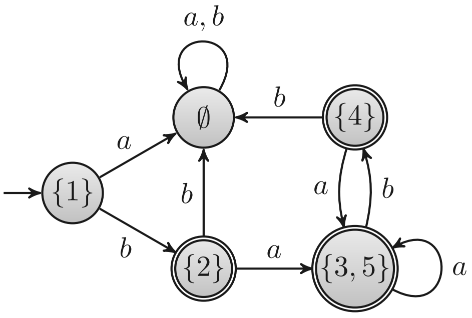
The proof of Proposition 3.2 would not work for arbitrary monoids M generated by Σ since the factorization of $u \in M \setminus \{1\}$ in terms of $u = u'a$ with $u' \in M$ and $a \in \Sigma$ might not be unique. Also note that we did not require that Σ are $\mathcal{A}$ are finite. If $\mathcal{A}$ has n states, then its powerset construction $\mathcal{P}(\mathcal{A})$ has at most 2^n states.

Theorem 3.3 (Kleene)**.** *For every finite alphabet Σ we have $\mathrm{REC}(\Sigma^*) = \mathrm{RAT}(\Sigma^*)$.*

Proof. The inclusion $\mathrm{REC}(\Sigma^*) \subseteq \mathrm{RAT}(\Sigma^*)$ follows from McKnight's Theorem 3.1. Consider a language $L \in \mathrm{RAT}(\Sigma^*)$. By Theorem 1.7 there exists a finite letter-by-letter Σ^*-automaton $\mathcal{A} = (Q, \delta, \{q_0\}, F)$ with $L(\mathcal{A}) = L$. The powerset construction $\mathcal{P}(\mathcal{A})$ has finitely many states, and it satisfies $L(\mathcal{P}(\mathcal{A})) = L$ by Proposition 3.2. By the Myhill-Nerode Theorem 2.7 we conclude $L \in \mathrm{REC}(\Sigma^*)$. □

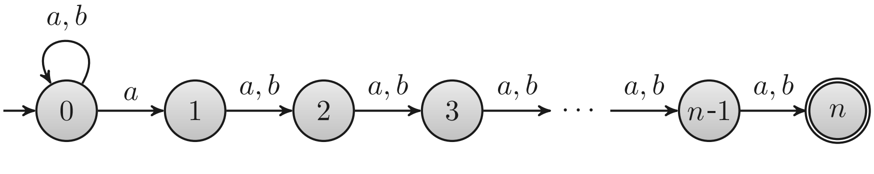
Example 3.1. Let $\Sigma = \{a, b\}$. The automaton on the left is the Σ^*-automaton from Example 1.9. On the right is its powerset construction.





Note that the empty set $\emptyset$ also occurs as a state in the powerset construction. Otherwise the powerset construction would not be complete. ◇

Example 3.2. Let $\Sigma = \{a, b\}$ and consider the language $L = \Sigma^* a \Sigma^{n-1}$. The following nondeterministic automaton with $n + 1$ states accepts L.



The powerset construction of the above automaton has 2^n states. For any word $u = a_n \cdots a_1$ with $a_i \in \Sigma$ we have $0 \xrightarrow{u} j$ for $j \in \{1, \ldots, n\}$ if and only if $a_j = a$. Therefore every word of length n uniquely determines a subset $P \subseteq \{1, \ldots, n\}$. Thus the powerset construction has at least 2^n states (given by the sets $P \cup \{0\}$). Since 0 is contained in all states of the powerset construction, the powerset construction cannot have more than 2^n states.

Consider two distinct states $P, P' \subseteq Q$ of the powerset construction. Without loss of generality, there exists $j \in P \setminus P'$. We have $1 \leq j \leq n$. Thus $n = j \cdot b^{n-j} \in P \cdot b^{n-j}$, but $n \notin P' \cdot b^{n-j}$. This shows that the powerset construction yields the minimal automaton of the language L. ◇

The above example shows that in the worst case, the powerset construction of an n-state automaton can have at least 2^{n-1} states. We refer to Example 4.6 for a proof the bound 2^n on the powerset construction of an n-state automaton is tight.

3.3. Nondeterministic automata and Boolean matrices

By the Myhill-Nerode Theorem 2.7 we have two equivalent characterizations of the recognizable languages, finite monoids and deterministic automata with finitely many states. The powerset construction translates a nondeterministic automaton with n states into a deterministic deterministic one with up to 2^n states. The transition monoid of a deterministic n-state automaton can have up to n^n elements. Therefore, when starting with a nondeterministic n-state automaton, using the powerset construction, and building its transition monoid, then the resulting upper bound for the size of this monoid is $(2^n)^{2^n} = 2^{n2^n}$. In this section we show that, instead of this doubly exponential bound, the singly exponential bound 2^{n^2} is sufficient.

There are two different views on the following result. First, it can be seen as a generalization of transition monoids to nondeterministic automata. And second, it can be interpreted as an implementation of nondeterministic automata using Boolean matrices.

Theorem 3.4. *Let Σ be an alphabet. If $L \subseteq \Sigma^*$ is accepted by a nondeterministic Σ^*-automaton $\mathcal{A} = (Q, \delta, I, F)$ with $|Q| = n$ and $\delta \subseteq Q \times \Sigma \times Q$, then L is recognized by a monoid with at most 2^{n^2} elements.*

Proof. We assume $Q = \{1, \ldots, n\}$. Let $\mathbb{B} = \{1, 0\}$ denote the Boolean lattice. The $n \times n$ Boolean matrices are denoted by $\mathbb{B}^{n \times n}$. There are 2^{n^2} Boolean matrices. The entry of the matrix $A \in \mathbb{B}^{n \times n}$ in row i and column j is written as A_{ij}. For every letter $a \in \Sigma$ we define the matrix $A^a \in \mathbb{B}^{n \times n}$ by

$$A^a_{ij} = 1 \Leftrightarrow (i, a, j) \in \delta.$$

This yields the homomorphism $\psi : \Sigma^* \to \mathbb{B}^{n \times n}$ with

$$\psi(a_1 \cdots a_n) = A^{a_1} \cdots A^{a_n}$$

for letters $a_i \in \Sigma$. The term $A^{a_1} \cdots A^{a_n}$ denotes the matrix product of $A^{a_1}, \ldots, A^{a_n}$ in which we use the bitwise *OR* (denoted by $\vee$) for addition, and the bitwise *AND* (denoted by $\wedge$) for multiplication. Suppose $\psi(u) = B$. We claim that

$$B_{ij} = 1 \Leftrightarrow i \xrightarrow{u} j.$$

The proof of the claim is by induction on the length of u. If $u = \varepsilon$, then B is the identity matrix and the claim is true since $\mathcal{A}$ has no ε-transitions. Let now $u = u'a$ for $a \in \Sigma$. By induction, we can assume that the matrix $B' = \psi(u')$ has the desired property for u'. We have $B = B'A^a$ and thus

$$\begin{aligned} B_{ij} = 1 &\Leftrightarrow \bigvee_{k=1}^{n} \left(B'_{ik} \wedge A^a_{kj}\right) \\ &\Leftrightarrow i \xrightarrow{u'} k \text{ and } (k,a,j) \in \delta \text{ for some } k \in \{1,\ldots,n\} \\ &\Leftrightarrow i \xrightarrow{u} j. \end{aligned}$$

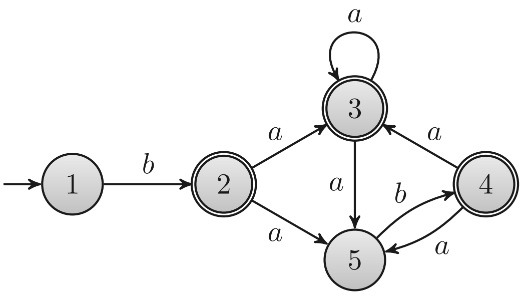
This proves the claim. We conclude $L = \psi^{-1}(P)$ for

$$P = \left\{ B \in \mathbb{B}^{n\times n} \;\middle|\; \exists i \in I\, \exists j \in F \colon B_{ij} = 1 \right\}.$$

This shows that ψ recognizes the language L. □

Several remarks regarding the proof of Theorem 3.4 are in order. First, note that when replacing Σ^* by some arbitrary Σ-generated monoid M, then ψ would not be well-defined in general: An element $u \in M$ could be written as both $u = a_1 \cdots a_n$ and $u = b_1 \cdots b_m$ which might result in two different matrices for u. Second, instead of considering all Boolean matrices one usually obtains considerably smaller monoids by only using the image $\psi(\Sigma^*) \subseteq \mathbb{B}^{n\times n}$. The third remark is that Boolean matrices can be used as a way of implementing nondeterministic automata; instead of handling the automaton as a labeled graph one could alternatively rely on the matrices $\{ A^a \in \mathbb{B}^{n\times n} \mid a \in \Sigma \}$ together with the initial states I and the final states F.

Example 3.3. Let $\Sigma = \{a, b\}$. We again consider the Σ^*-automaton from Example 1.9.



We have

$$A^a = \begin{pmatrix} 0&0&0&0&0\\ 0&0&1&0&1\\ 0&0&1&0&1\\ 0&0&1&0&1\\ 0&0&0&0&0 \end{pmatrix} \qquad A^b = \begin{pmatrix} 0&1&0&0&0\\ 0&0&0&0&0\\ 0&0&0&0&0\\ 0&0&0&0&0\\ 0&0&0&1&0 \end{pmatrix}$$

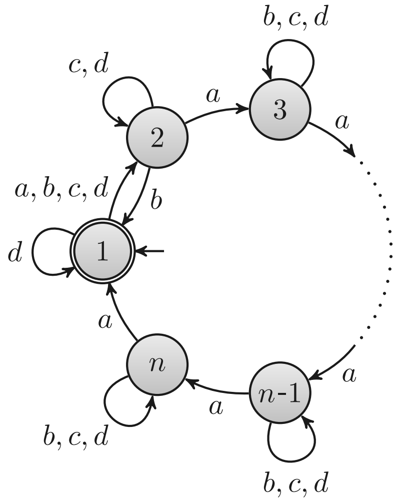
By matrix multiplication one obtains for instance the following matrices.

$$A^bA^a = \begin{pmatrix} 0&0&1&0&1\\ 0&0&0&0&0\\ 0&0&0&0&0\\ 0&0&0&0&0\\ 0&0&1&0&1 \end{pmatrix} \quad A^aA^b = \begin{pmatrix} 0&0&0&0&0\\ 0&0&0&1&0\\ 0&0&0&1&0\\ 0&0&0&1&0\\ 0&0&0&0&0 \end{pmatrix} \quad A^bA^aA^aA^b = \begin{pmatrix} 0&0&0&1&0\\ 0&0&0&0&0\\ 0&0&0&0&0\\ 0&0&0&0&0\\ 0&0&0&1&0 \end{pmatrix}$$

Note that the matrix $B = A^bA^aA^aA^b$ was obtained by multiplication of the two matrices A^bA^a and A^aA^b. The entry 1 of the matrix B in row 1, column 4 shows that *baab* is accepted by the automaton since it indicates a run from the initial state 1 to the final state 4. ◇

The following example shows that the bound 2^{n^2} in Theorem 3.4 is tight.

Example 3.4. Let $\Sigma = \{a, b, c, d\}$. We extend the automaton in Example 2.5 by some nondeterministic action d. This leads to the following n-state automaton $\mathcal{A}$.



Let $L = L(\mathcal{A})$. The automaton $\mathcal{A}$ generates all binary relations $R \subseteq \{1, \dots, n\}^2$; that is, for every such relation R there exists a word $u_R \in \Sigma^*$ such that $i \xrightarrow{u_R} j$ if and only if $(i, j) \in R$. There are 2^{n^2} binary relations on $\{1, \dots, n\}^2$. Therefore, the monoid M constructed in the proof of Theorem 3.4 has size 2^{n^2}. For showing that this bound cannot be improved, we verify that the syntactic congruence $\equiv_L$ of the language L satisfies $u_R \not\equiv_L u_S$ for all different relations $R, S \subseteq \{1, \dots, n\}^2$. Without loss of generality, we can assume $(i, j) \in R \setminus S$. Then $a^{i-1} u_R a^j \in L$ but $a^{i-1} u_S a^j \notin L$. Hence $u_R \not\equiv_L u_S$ as desired. ◇

Remark 3.1. An immediate consequence of Theorem 3.4 is another proof of Kleene's Theorem 3.3, which states that $\mathrm{REC}(\Sigma^*) = \mathrm{RAT}(\Sigma^*)$ for finite alphabets Σ. The inclusion from left to right is McKnight's Theorem 3.1. Let now $L \in \mathrm{RAT}(\Sigma^*)$. By Theorem 1.7 there exists a finite letter-by-letter automaton accepting L. By Theorem 3.4, the language L is recognized by a finite monoid, *i.e.*, we have $L \in \mathrm{REC}(\Sigma^*)$. ◇

3.4. The relation between rational and recognizable sets

We have more tools at hand for proving that sets are not recognizable than for proving that sets are not rational. For showing that a given set is not recognizable it suffices to show that its syntactic monoid is infinite. Kleene's Theorem therefore also yields an additional method for showing that some given subset (of a finitely generated free monoid) is not rational, simply by showing that it is not recognizable. We apply this technique in the next example for showing that, in general, rationals sets are not closed under intersection.

Example 3.5. Consider the monoid $M = \{a, b\}^* \times c^*$. We define the following rational subsets of M:

$$K_1 = \{(a, c),\ (b, \varepsilon)\}^*$$
$$K_2 = \{(a, \varepsilon),\ (b, c)\}^* \,.$$

The set K_1 consists of the elements (u, c^n) such that the word $u \in \{a, b\}^*$ has exactly n occurrences of the letter a. Similarly, K_2 contains those elements (u, c^n) such that u has exactly n occurrences of the letter b. We want to show that $K_1 \cap K_2$ is not rational. By contradiction, we assume that $K_1 \cap K_2$ is rational. We have

$$K_1 \cap K_2 = \{\, (u, c^n) \mid u \text{ has } n \text{ occurrences of the letter } a \text{ and } n \text{ occurrences of } b \,\} \,.$$

Consider the homomorphism $\psi : M \to \{a, b\}^*$ defined by $\psi(u, c^n) = u$. Since rational sets are closed under homomorphic images (Proposition 1.3), the language $L = \psi(K_1 \cap K_2) \subseteq \{a, b\}^*$ is rational. Note that

$$L = \{\, u \in \{a, b\}^* \mid u \text{ has the same number of } a\text{'s as } b\text{'s} \,\} \,.$$

	rational sets	**recognizable sets**	**regular languages**
union:	✔ (by definition)	✔ (Theorem 2.3)	✔
intersection:	✕ (Example 3.5)	✔ (Theorem 2.3)	✔
complement:	✕ (Example 3.5)	✔ (Theorem 2.3)	✔
residuals:	✕ (Example 1.4)	✔ (Theorem 2.3)	✔
inverse homomorphisms:	✕ (Example 1.3)	✔ (Theorem 2.3)	✔
homomorphisms:	✔ (Proposition 1.3)	✕ (Example 2.1)	✔
product:	✔ (by definition)	✕ (Example 2.2)	✔
Kleene star:	✔ (by definition)	✕ (Example 2.2)	✔

Table 3.1.: Closure properties of rational sets, recognizable sets, and regular languages.

The syntactic monoid of L is infinite. For instance, we have $a^m \not\equiv_L a^n$ for $m \neq n$ since $a^m b^m \in L$ and $a^n b^m \notin L$. This shows that L is not recognizable, and by Kleene's Theorem L is not rational. This is a contradiction. Therefore, the assumption is wrong and $K_1 \cap K_2$ is not rational. In particular, $\mathrm{RAT}(M)$ is not closed under intersection. Moreover, $\mathrm{RAT}(M)$ cannot be closed under complement since otherwise by DeMorgan's law it would also be closed under intersection (by definition $\mathrm{RAT}(M)$ is closed unter union). ◇

We have seen in Example 1.3 that in general $\mathrm{RAT}(M)$ is not closed under inverse homomorphism. The example used non-finitely generated monoids. The following example shows that $\mathrm{RAT}(M)$ is not closed under inverse homomorphisms even if M is finitely generated.

Example 3.6. Let $M = \mathbb{N} \times \mathbb{N}$ with componentwise addition and consider the set $\{(1,1)\}^* = \{\, (n,n) \mid n \in \mathbb{N} \,\}$. The mapping $\psi : \{a,b\}^* \to N$ with $\psi(a) = (1,0)$ and $\psi(b) = (0,1)$ defines a homomorphism. We have $\psi^{-1}(L) = \{\, u \in \{a,b\}^* \mid u \text{ has the same number of } a\text{'s as } b\text{'s} \,\}$ which is not rational (as we have seen in Example 3.5. ◇

Table 3.1 summarizes the closure properties of rational sets, recognizable sets, and regular languages. By Kleene's Theorem, the regular languages have the closure properties of both the rational and the recognizable sets.

For regular languages we can combine Figures 1.1 and 2.1 into Figure 3.1. The terms after the names of the constructions indicate the maximal blop-up relative to the number of states, the size of expressions, and the size of monoids. As in Figure 1.1, we assume that the set of generators Σ is fixed.

Recognizable and rational sets in direct products

Suppose $M = M_1 \times M_2$. We are interested in the relation between $\mathrm{REC}(M)$ and $\mathrm{REC}(M_i)$ for $i \in \{1,2\}$. As it turns out, $\mathrm{REC}(M)$ is fully determined by $\mathrm{REC}(M_1)$ and $\mathrm{REC}(M_2)$.

Theorem 3.5 (Mezei)**.** *Let M_1 and M_2 be monoids. Then $L \in \mathrm{REC}(M_1 \times M_2)$ if and only if L is a finite union of sets of the form $K_1 \times K_2$ for $K_1 \in \mathrm{REC}(M_1)$ and $K_2 \in \mathrm{REC}(M_2)$.*

Proof. $\Rightarrow$: Let $\varphi : M_1 \times M_2 \to N$ be a homomorphism to a finite monoid N which recognizes the set L. We define homomorphisms $\psi_1 : M_1 \to N$ and $\psi_2 : M_2 \to N$ by $\psi_1(m_1) = \varphi(m_1, 1)$ and $\psi_2(m_2) = \varphi(1, m_2)$. This yields the homomorphism $\psi : M_1 \times M_2 \to N \times N$ with $\psi(m_1, m_2) = (\psi_1(m_1), \psi_2(m_2))$. The homomorphism ψ recognizes L since the set $P = \{\, (n_1, n_2) \in N \times N \mid n_1 n_2 \in \varphi(L) \,\}$ satisfies

$$\begin{aligned}
\psi^{-1}(P) &= \{\, (m_1, m_2) \mid \psi(m_1, m_2) \in P \,\} \\
&= \{\, (m_1, m_2) \mid \psi_1(m_1)\, \psi_2(m_2) \in \varphi(L) \,\} \\
&= \{\, (m_1, m_2) \mid \varphi(m_1, 1)\, \varphi(1, m_2) \in \varphi(L) \,\} \\
&= \{\, (m_1, m_2) \mid \varphi(m_1, m_2) \in \varphi(L) \,\} = \varphi^{-1}\big(\varphi(L)\big) = L.
\end{aligned}$$

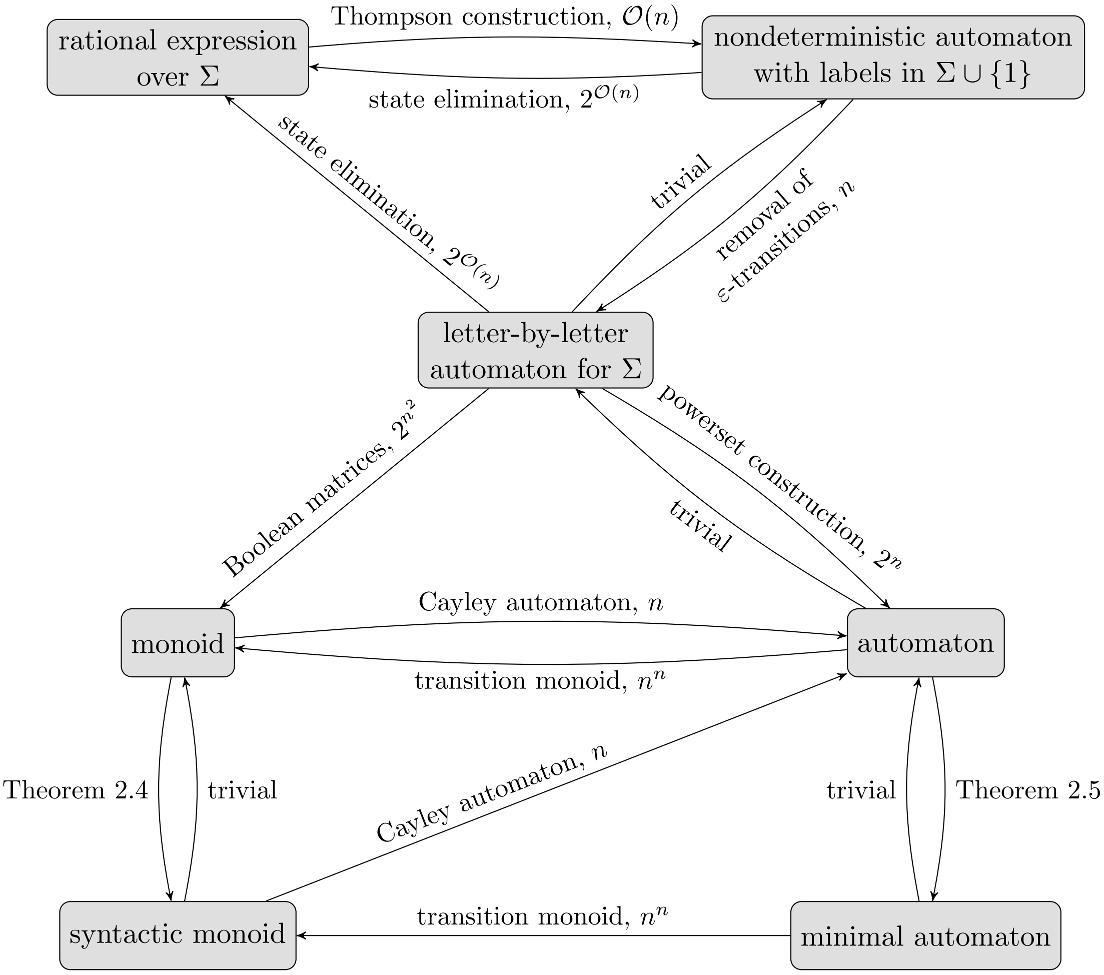

rational expression over Σ
Thompson construction, $\mathcal{O}(n)$
nondeterministic automaton with labels in $\Sigma \cup \{1\}$
state elimination, $2^{\mathcal{O}(n)}$
state elimination, $2^{\mathcal{O}(n)}$
trivial
removal of ε-transitions, n
letter-by-letter automaton for Σ
Boolean matrices, 2^{n^2}
powerset construction, 2^n
trivial
Cayley automaton, n
monoid
automaton
transition monoid, n^n
Theorem 2.4
trivial
Cayley automaton, n
trivial
Theorem 2.5
syntactic monoid
transition monoid, n^n
minimal automaton

Figure 3.1.: Transformations between descriptions of regular languages.

We conclude

$$L = \psi^{-1}\big(\psi(L)\big) \;=\; \bigcup_{(n_1,n_2)\in\psi(L)} \psi^{-1}(n_1,n_2) \;=\; \bigcup_{(n_1,n_2)\in\psi(L)} \psi_1^{-1}(n_1) \times \psi_2^{-1}(n_2)$$

showing that L has the desired form.

$\Leftarrow$: Let $L = K_1 \times K_2$, and for $i \in \{1,2\}$ let $\varphi_i : M_i \to N_i$ to a finite monoid N_i which recognizes K_i. We define $\psi : M_1 \times M_2 \to N_1 \times N_2$ by $\psi(m_1,m_2) = (\varphi_1(m_1), \varphi_2(m_2))$. Then

$$\begin{aligned}\psi^{-1}\big(\psi(L)\big) &= \psi^{-1}\big(\varphi_1(K_1) \times \varphi_2(K_2)\big) \\ &= \varphi_1^{-1}\big(\varphi_1(K_1)\big) \times \varphi_2^{-1}\big(\varphi_2(K_2)\big) = K_1 \times K_2 = L.\end{aligned}$$

Thus $L \in \mathrm{REC}(M_1 \times M_2)$. The implication from right to left now follows from the fact that $\mathrm{REC}(M_1 \times M_2)$ is closed under union. □

The following example shows that the implication from left to right in Mezei's Theorem 3.5 becomes false when replacing REC by RAT.

Example 3.7. Consider $M = a^* \times b^*$. The subset $L = (a,b)^* = \{\, (a^n, b^n) \mid n \geq 0 \,\}$ is rational. Assume L can be written as a finite union of sets of the form $K_1 \times K_2$ with $K_1 \in \mathrm{RAT}(a^*)$ and $K_2 \in \mathrm{RAT}(b^*)$. By Kleene's Theorem, the languages K_i are recognizable and by Mezei's Theorem L is recognizable. This is a contradiction since the syntactic monoid of L is infinite (and isomorphic to $\mathbb{Z}$). Therefore, L has no presentation of the above form. ◇

The implication from right to left in Mezei's Theorem can be extended to rational sets.

Proposition 3.6. *Let M_1 and M_2 be monoids. If $K_1 \in \mathrm{RAT}(M_1)$ and $K_2 \in RAT(M_2)$, then $K_1 \times K_2 \in \mathrm{RAT}(M_1 \times M_2)$.*

Proof. Let s_i be a rational expression for K_i. We replace every atom $a \in M_1$ in the expression s_1 by $(a,1)$. This yields the expression t_1. Symmetrically, we replace every atom $b \in M_2$ in the expression s_2 by $(1,b)$ which yields t_2. We have $u \in L(s_1)$ if and only if $(u,1) \in L(t_1) \subseteq M_1 \times M_2$. Similarly, $v \in L(s_2)$ if and only if $(1,v) \in L(t_2)$. Thus the concatenation t_1t_2 defines $K_1 \times K_2$. □

Even though $\mathrm{REC}(M)$ is not closed under product in general, the following consequence of Mezei's Theorem shows that direct products preserve this closure property.

Proposition 3.7. *Let M_1 and M_2 be monoids such that both $\mathrm{REC}(M_1)$ and $\mathrm{REC}(M_2)$ are closed under product. Then $\mathrm{REC}(M_1 \times M_2)$ is closed under product, too.*

Proof. Let $K, L \in \mathrm{REC}(M_1 \times M_2)$. By Mezei's Theorem, we can write $K = \bigcup_k K_{1,k} \times K_{2,k}$ and $L = \bigcup_\ell L_{1,\ell} \times L_{2,\ell}$ for languages $K_{i,k}, L_{i,\ell} \in \mathrm{REC}(M_i)$ and each of the unions is finite. Since $\mathrm{REC}(M_i)$ is closed under product, we have $K_{i,k}L_{i,\ell} \in \mathrm{REC}(M_i)$. Again by Mezei's Theorem we see that the products $(K_{1,k} \times K_{2,k})(L_{1,\ell} \times L_{2,\ell}) = K_{1,k}L_{1,\ell} \times K_{2,k}L_{2,\ell}$ are in $\mathrm{REC}(M_1 \times M_2)$. This yields

$$KL = \bigcup_{k,\ell} \big(K_{1,k}L_{1,\ell} \times K_{2,k}L_{2,\ell}\big) \in \mathrm{REC}(M_1 \times M_2).$$ □

In particular, by Kleene's Theorem and Proposition 3.7, for finite alphabets Σ_i we see that $\mathrm{REC}(\Sigma_1^* \times \cdots \times \Sigma_n^*)$ is closed under product. We note that if both $\mathrm{REC}(M_1)$ and $\mathrm{REC}(M_2)$ are closed under Kleene star, then $\mathrm{REC}(M_1 \times M_2)$ does not have to be closed under Kleene star. For example, we have $\{(a,b)\} \in \mathrm{REC}(a^* \times b^*)$ but $(a,b)^*$ is not recognizable.

Intersections of rational and reconizable sets

We have seen in Example 3.5 that rational sets are not closed under intersection. Under this aspect, it is surprising that the intersection of a rational set and a recognizable set is rational again.

Proposition 3.8. *Let M be a monoid, let $K \in \mathrm{RAT}(M)$, and let $L \in \mathrm{REC}(M)$. Then $K \cap L \in \mathrm{RAT}(M)$.*

Proof. By Proposition 1.1, the set K is contained in a finitely generated submonoid M'. Let $M' \subseteq M$ be generated by the finite set Σ. The inclusion $\Sigma \subseteq M'$ induces a homomorphism $\psi : \Sigma^* \to M$ (with $\psi(a) = a$ for all $a \in \Sigma$). Note that Σ^* denotes the free monoid over Σ. Since the restriction $\psi : \Sigma^* \to M'$ is surjective, by Proposition 1.3 there exists $K' \in \mathrm{RAT}(\Sigma^*)$ with $\psi(K') = K$. Let $L' = \psi^{-1}(L)$. Theorem 2.3 shows $L' \in \mathrm{REC}(\Sigma^*)$. By Kleene's Theorem, both K' and L' are regular. Hence $K' \cap L' \in \mathrm{RAT}(\Sigma^*)$, and by Proposition 1.3 we obtain $\psi(K' \cap L') \in \mathrm{RAT}(M)$. We obtain

$$\begin{aligned}
\psi(K' \cap L') &= \psi\bigl(K' \cap \psi^{-1}(L)\bigr) \\
&= \psi\bigl(K' \cap \psi^{-1}(L \cap M')\bigr) \\
&= \psi(K') \cap (L \cap M') \\
&= K \cap L \cap M' = K \cap L.
\end{aligned}$$

The third equality holds since the restriction $\psi : \Sigma^* \to M'$ is surjective. □

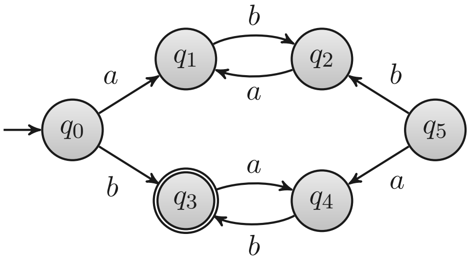
4. Algorithmic Properties of Automata

Regular languages have a great number of important closure properties. One reason for the success of the regular languages is that these closure properties are effective. This means that for given automata one can actually compute an automaton for the union (resp. intersection, complement, residual, homomorphism, inverse homomorphism, concatenation, Kleene star). In addition, for many decision problems such as emptiness, universality, inclusion, or equivalence there exist efficient algorithms when the input languages are given as deterministic automata.

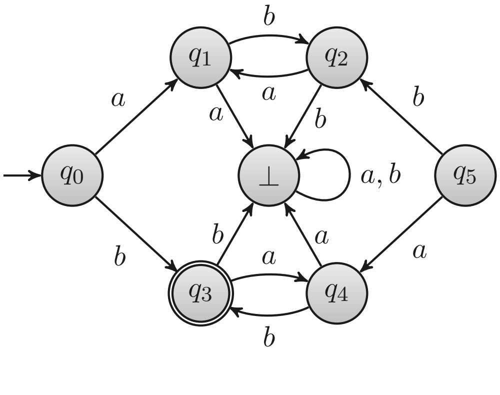
As operations in an automaton (Q, δ, I, F) with $\delta \subseteq Q \times \Sigma \times Q$ we consider membership queries of the form "Is state q in F?" or "Is (p, a, q) a transition in δ?". Moreover, we assume that we can efficiently enumerate all states in Q, I, or F as well as all transitions in δ. In the case of deterministic automata, the computation of the transition function $q \cdot a$ is a typical operation.

In some situations it is important that automata are "deadlock-free". This is formalized by the following notion. An automaton $\mathcal{A} = (Q, \delta, I, F)$ with $\delta \subseteq Q \times \Sigma \times Q$ is *complete* if for every state $p \in Q$ and every letter $a \in \Sigma$ there exists a transition of the form $(p, a, q) \in \delta$, *i.e.*, we always have an outgoing transition for every letter. An automaton can always be made complete by adding a new state and redirecting all missing transtions to this new state. By definition, deterministic automata are complete.

Example 4.1. Let $\mathcal{A}$ be the following $\{a, b\}^*$-automaton for $L = b(ab)^*$.



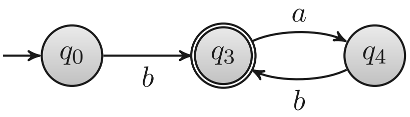
It is not complete since, for instance, there is no outgoing a-transition in state q_1. An equivalent complete automaton can be obtained by adding an new sink state $\perp$ and introducing additional transitions as depicted below.



This complete automaton contains several "unproductive" states which do not contribute to accepting runs. In particular the state $\perp$ is unproductive in this sense. ◇

The counterpart of being complete is being trim. A state which is not reachable from an initial state or which cannot reach a final state cannot occur on an accepting run. Thus removing such a state does not change the accepted language. This leads to the following definition. A Σ^*-automaton $\mathcal{A} = (Q, \delta, I, F)$ is *trim* if for every state p there exist words $u, v \in \Sigma^*$, an initial state $i \in I$ and a final state $f \in F$ such that $i \xrightarrow{u} p \xrightarrow{v} f$. Every automaton can be trimmed by removing all states which either are unreachable from an initial state or which cannot reach a final state.

Example 4.2. The automaton $\mathcal{A}$ in Example 4.1 is not trim. When trimming $\mathcal{A}$ (or equivalently its complete counterpart) we obtain the following automaton.



Note that this automaton is not complete. ◇

The automaton $\mathcal{A}$ in Example 4.1 is neither complete nor trim. Of course there also exist automata which are both complete and trim at the same time. The simplest example is the one state automaton for the language $L = \Sigma^*$.

4.1. Boolean operations

Complementing automata

Complementing an automaton over Σ^* is the computation of the following function.

Input: A finite Σ^*-automaton $\mathcal{A}$.

Output: A finite automaton $\mathcal{B}$ with $L(\mathcal{B}) = \Sigma^* \setminus L(\mathcal{A})$.

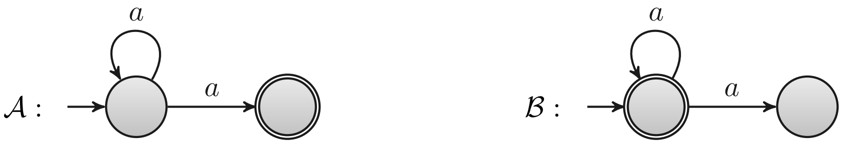
Suppose the given automaton $\mathcal{A} = (Q, \cdot, q_0, F)$ is deterministic. Its *complement automaton* $\overline{\mathcal{A}}$ is defined as

$$\overline{\mathcal{A}} = (Q, \cdot, q_0, Q \setminus F),$$

and we have $L(\overline{\mathcal{A}}) = \Sigma^* \setminus L(\mathcal{A})$. Note that this construction works for deterministic automata over arbitrary monoids. Depending on the implementation of $\mathcal{A}$, computing the complement automaton usually is linear (or even constant with an adequate data structure for sets of states; for instance, one could use an additional bit indicating whether or not to invert the output of a membership query).

In general, $\mathrm{RAT}(M)$ is not closed under complement. Therefore, this is a strong indication that when $\mathcal{A}$ is nondeterministic, then applying Kleene's Theorem in some form cannot really be avoided. In particular, for complementing a nondeterministic automaton over Σ^*, we compute the complement automaton of the powerset construction of $\mathcal{A}$. The following example shows that simply interchanging final and non-final states does not work for nondeterministic automata.

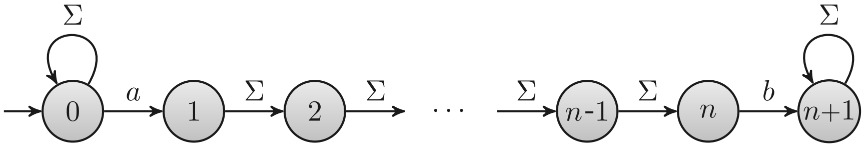
Example 4.3. Consider the following two nondeterministic a^*-automata.



The automaton $\mathcal{B}$ is obtain from $\mathcal{A}$ by interchanging final and non-final states. We have $L(\mathcal{A}) = a^*a$ and $L(\mathcal{B}) = a^*$. In particular, $L(\mathcal{A})$ is a subset of $L(\mathcal{B})$ whereas the complement is $a^* \setminus L(\mathcal{A}) = \{\varepsilon\}$. ◇

Complementing the powerset construction can yield a deterministic automaton of exponential size. The following example is due to Holzer and Kutrib [?]; it shows that even for nondeterministic outputs, one cannot avoid the exponential blow-up.

Example 4.4. Let $\Sigma = \{a, b\}$. We show that for every $n \geq 1$ there exists a language $L \subseteq \Sigma^*$ accepted by a nondeterministic automaton with $n + 2$ states such that every letter-by-letter automaton for the complement of L has at least $2^n - 1$ states. Let $L = \Sigma^* a \Sigma^{n-1} b \Sigma^*$. The language L is accepted by the following nondeterministic automaton with $n + 2$ states.



Let $\mathcal{B} = (Q, \delta, I, F)$ be a nondeterministic automaton with $\delta \subseteq Q \times \Sigma \times Q$ accepting $\Sigma^* \setminus L$. We consider words in $K = \Sigma^n \setminus \{b^n\}$. For every word $u \in K$ we have $uu \in L(\mathcal{B})$. Thus for every word $u \in K$ there exist states $q_{0,u} \in I$, $q_u \in Q$, and $q_{1,u} \in F$ such that $q_{0,u} \xrightarrow{u} q_u \xrightarrow{u} q_{1,u}$. Suppose $q_u = q_v$ for two different words $u, v \in K$. By looking at the first difference between u and v, we see that there exists a position i such that one of the following holds:

(a) Either the i-th letter in u is a and the i-th letter of v is b,

(b) or the i-th letter in u is b and the i-th letter of v is a.

We can assume that (a) holds (since otherwise we interchange u and v). In particular, $uv \in L$. We have $q_{0,u} \xrightarrow{u} q_u = q_v \xrightarrow{v} q_{1,v}$ and thus $uv \in L(\mathcal{B})$. This is a contradiction. Therefore, for different words $u, v \in K$ we have $q_u \neq q_v$, which yields

$$2^n - 1 = |K| = |\{\, q_u \in Q \mid u \in K \,\}| \leq |Q| \,.$$

This also shows that the minimal deterministic automaton of L has at least $2^n - 1$ states since the minimal automaton of L and the minimal automaton of its complement $\Sigma^* \setminus L$ only differ in the final states. ◇

The union operation for automata

We consider the following problem for automata over Σ^*.

Input: Two finite Σ^*-automata $\mathcal{A}_1$ and $\mathcal{A}_2$.

Output: A finite automaton $\mathcal{B}$ with $L(\mathcal{B}) = L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$.

Let $\mathcal{A}_1 = (Q_1, \delta_1, I_1, F_1)$ and $\mathcal{A}_2 = (Q_2, \delta_2, I_2, F_2)$ with $Q_1 \cap Q_2 = \emptyset$. There are two different constructions for the union of $L(\mathcal{A}_1)$ and $L(\mathcal{A}_2)$. The first construction is the *union automaton*

$$\mathcal{A}_1 \cup \mathcal{A}_2 = (Q_1 \cup Q_2, \delta_1 \cup \delta_2, I_1 \cup I_2, F_1 \cup F_2).$$

We trivially have $L(\mathcal{A}_1 \cup \mathcal{A}_2) = L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$. With $|Q_1| + |Q_1|$ states, the size of the union automaton is linear. Also note that the union automaton construction works for automata over arbitrary monoids. The union automaton $\mathcal{A}_1 \cup \mathcal{A}_2$ is nondeterministic even if $\mathcal{A}_1$ and $\mathcal{A}_2$ are deterministic.

Next, we consider product automata. This construction yields bigger automata but it preserves determinism. Moreover, by an appropriate choice of the final states it can be used for other Boolean operations, too. If $\delta_i \subseteq Q_i \times \Sigma \times Q_i$ for $i \in \{1, 2\}$, then one can also use the *product automaton*

$$\mathcal{A}_1 \times \mathcal{A}_2 = (Q_1 \times Q_2, \delta, I_1 \times I_2, F)$$

where $\big((p_1, p_2), a, (q_1, q_2)\big) \in \delta$ if both $(p_1, a, q_1) \in \delta_1$ and $(p_2, a, q_2) \in \delta_2$. It is often convenient to write a state $(p, q) \in Q_1 \times Q_2$ as $\left[\begin{smallmatrix} p \\ q \end{smallmatrix}\right]$. There exists a run

$$\begin{bmatrix} p_1 \\ q_1 \end{bmatrix} \xrightarrow{a_1} \begin{bmatrix} p_2 \\ q_2 \end{bmatrix} \xrightarrow{a_2} \begin{bmatrix} p_3 \\ q_3 \end{bmatrix} \xrightarrow{a_3} \cdots \xrightarrow{a_{n-1}} \begin{bmatrix} p_n \\ q_n \end{bmatrix} \xrightarrow{a_n} \begin{bmatrix} p_{n+1} \\ q_{n+1} \end{bmatrix}$$

in the product automaton $\mathcal{A}_1 \times \mathcal{A}_2$ if and only if there exist runs

$$p_1 \xrightarrow{a_1} p_2 \xrightarrow{a_2} p_3 \xrightarrow{a_3} \cdots \xrightarrow{a_{n-1}} p_n \xrightarrow{a_n} p_{n+1} \quad \text{in } \mathcal{A}_1,$$
$$q_1 \xrightarrow{a_1} q_2 \xrightarrow{a_2} q_3 \xrightarrow{a_3} \cdots \xrightarrow{a_{n-1}} q_n \xrightarrow{a_n} q_{n+1} \quad \text{in } \mathcal{A}_2.$$

For the union $L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$ of complete automata $\mathcal{A}_1$ and $\mathcal{A}_2$, we can use the product automaton construction with final states $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$. If $\mathcal{A}_1 = (Q_1, \cdot, q_{01}, F_1)$ and $\mathcal{A}_2 = (Q_2, \cdot, q_{02}, F_2)$ are given as deterministic automata, then the transistion function $\cdot : Q_1 \times Q_2 \times \Sigma^* \to Q_1 \times Q_2$ of the product automaton is defined componentwise by $(p, q) \cdot u = (p \cdot u,\, q \cdot u)$, and the initial state is (q_{01}, q_{02}). The product automaton construction over nondeterministic automata only works for free monoids whereas over deterministic automata, one can apply this construction for arbitrary monoids. The size of the product automaton is quadratic. As usual, it suffices to consider reachable states, only. This can result in smaller automata.

Intersections of regular languages

There exist monoids M such that the rational subsets of M are not closed under intersection, see Example 3.5. Therefore, we only consider the intersection of regular languages. The *intersection problem* is the following.

Input: Two finite Σ^*-automata $\mathcal{A}_1$ and $\mathcal{A}_2$.

Output: A finite automaton $\mathcal{B}$ with $L(\mathcal{B}) = L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$.

Let $\mathcal{A}_i = (Q_i, \delta_i, I_i, F_i)$ with $\delta_i \subseteq Q_i \times \Sigma \times Q_i$. Let $\mathcal{B}$ be the product automaton $\mathcal{A}_1 \times \mathcal{A}_2$ with final states $F_1 \times F_2$. Then $L(\mathcal{B})$ accepts $L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$, even if $\mathcal{A}_1$ and $\mathcal{A}_2$ are not complete. Note that in the case of deterministic automata $\mathcal{A}_1$ and $\mathcal{A}_2$, the product automaton $\mathcal{A}_1 \times \mathcal{A}_2$ can accept any Boolean combination of $L(\mathcal{A}_1)$ and $L(\mathcal{A}_2)$, depending on the choice of the final states.

4.2. Homomorphisms and inverse homomorphisms

Homomorphic images

The computation of homomorphic images of automata is the following problem.

Input: A finite Σ^*-automaton $\mathcal{A}$ and a homomorphism $\psi : \Sigma^* \to \Gamma^*$.

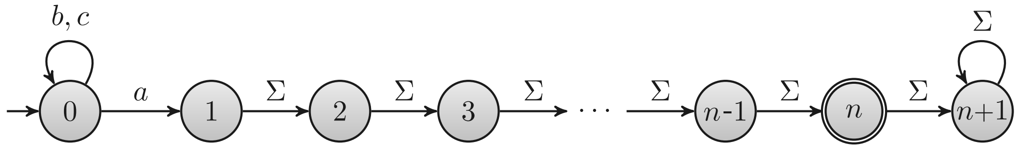
Output: A finite Γ^*-automaton $\mathcal{B}$ with $L(\mathcal{B}) = \psi(L(\mathcal{A}))$.

As usual, $\psi(L) = \{\, \psi(u) \mid u \in L \,\}$. We mimic the proof of Proposition 1.3. For a nondeterministic automaton $\mathcal{A} = (Q, \delta, I, F)$ we set $\mathcal{B} = (Q, \delta', I, F)$ with

$$\delta' = \left\{\, \big(p, \psi(u), q\big) \;\middle|\; (p, u, q) \in \delta \,\right\}.$$

Now, if $q_1 u_1 q_2 \cdots u_n q_{n+1}$ is a run of $\mathcal{A}$ on u if and only if $q_1\, \psi(u_1)\, q_2 \cdots \psi(u_n)\, q_{n+1}$ is a run of $\mathcal{B}$ on $\psi(u)$. Thus $L(\mathcal{B}) = \psi(L(\mathcal{A}))$ as desired. If $\delta \subseteq Q \times \Sigma \times Q$, then, by replacing a single transition by many as in the proof of Theorem 1.6, we obtain an automaton $\tilde{\mathcal{B}}$ with transition relation $\tilde{\delta} \subseteq Q \times \Gamma \times Q$. If $n = |Q|$ and $m = \max |\psi(a)| a \in \Sigma$, then $\tilde{\mathcal{B}}$ has at most nm states. If $\mathcal{A}$ is deterministic, then $\mathcal{B}$ does not have to be deterministic. Moreover, even if $\psi(a) \in \Gamma$ for all $a \in \Sigma$, the minimal automaton of $\psi(L(\mathcal{A}))$ for a deterministice automaton $\mathcal{A}$ can have exponential size.

Example 4.5. Let $\Gamma = \{a, b\}$, let $\Sigma = \{a, b, c\}$, and consider the following deterministic Σ^*-automaton $\mathcal{A}$ with $n + 2$ states.



Let $\psi : \Sigma^* \to \Gamma^*$ be the homomorphism defined by $\psi(a) = \psi(c) = a$ and $\psi(b) = b$. Then $\psi(L(\mathcal{A})) \subseteq \{a, b\}^*$ is the language from Example 3.2. Since its minimal deterministic automaton has 2^n states, this shows that one cannot avoid the exponential blow-up when computing deterministic automata for homomorphic images, even if the input automaton is deterministic. ◇

Inverse homomorphisms

We consider the following problem.

Input: A finite Σ^*-automaton $\mathcal{A}$ and a homomorphism $\psi : \Gamma^* \to \Sigma^*$.

Output: A finite Γ^*-automaton $\mathcal{B}$ with $L(\mathcal{B}) = \psi^{-1}(L(\mathcal{A}))$.

As usual, $\psi^{-1}(L) = \{\, u \mid \psi(u) \in L \,\}$. Let $\mathcal{A} = (Q, \delta, I, F)$ be a nondeterministic automaton with $\delta \subseteq Q \times \Sigma \times Q$. We set $\mathcal{B} = (Q, \delta', I, F)$ with

$$\delta' = \left\{ (p, a, q) \in Q \times \Gamma \times Q \;\middle|\; p \xrightarrow{\psi(a)} q \text{ in } \mathcal{A} \right\}.$$

Let $u = a_1 \cdots a_n$ with $a_i \in \Gamma$. There exists a run $q_1 a_1 q_2 \cdots a_n q_{n+1}$ in $\mathcal{B}$ if and only if we have

$$q_1 \xrightarrow{\psi(a_1)} q_2 \xrightarrow{\psi(a_2)} \cdots \; q_n \xrightarrow{\psi(a_2)} q_{n+1}$$

in $\mathcal{A}$. The latter condition is equivalent to $q_1 \xrightarrow{\psi(u)} q_{n+1}$ in $\mathcal{A}$. Note that the implication from right to left of this equivalence would not hold if we had labels of length greater than 1; moreover, this implication does not hold for automata over arbitrary monoids. We conclude that $L(\mathcal{B}) = \psi^{-1}(L(\mathcal{A}))$ as desired.

If $\mathcal{A} = (Q, \cdot, q_0, F)$ is a deterministic Σ^*-automaton, then translating the above construction from transition relations to transition functions yields the Γ^*-automaton $\mathcal{B} = (Q, \circ, q_0, F)$ with $q \circ u = q \cdot \psi(u)$. In particular, $\mathcal{B}$ is deterministic, too.

4.3. Residuals and quotients

We have already seen in Theorem 2.3 that regular languages are closed under residuals. Quotients are generalizations of residuals. In this section we give constructions on automata for computing residuals and quotients.

Residuals

The languages $u^{-1}L = \{\, v \in \Sigma^* \mid uv \in L \,\}$ is called the *left residual* of $L \subseteq \Sigma^*$ by the word $u \in \Sigma^*$. Symmetrically, $Lu^{-1} = \{\, v \in \Sigma^* \mid vu \in L \,\}$ is a *right residual.* We say that K is a *residual* of L if it is either a left or a right residual. Computing residuals is the following problem.

Input: A finite Σ^*-automaton $\mathcal{A}$ and a word $u \in \Sigma^*$.

Output: A finite Σ^*-automaton $\mathcal{B}$ with $L(\mathcal{B}) = u^{-1}L(\mathcal{A})$ in the case of left residuals (or with $L(\mathcal{B}) = L(\mathcal{A})u^{-1}$ in the case of right residuals).

We first consider left residuals. If $\mathcal{A} = (Q, \delta, I, F)$ is a nondeterministic automaton with $\delta \subseteq Q \times \Sigma \times Q$, we set $\mathcal{B} = (Q, \delta, I', F)$ with

$$I' = \left\{\, q \in Q \;\middle|\; \exists p \in I \colon p \xrightarrow{u} q \,\right\}.$$

Let $r \in Q$ and $v \in \Sigma^*$. Then the following conditions are equivalent:

- There exists $q \in I'$ with $q \xrightarrow{v} r$.
- There exist $q \in I'$ and $p \in I$ with $p \xrightarrow{u} q \xrightarrow{v} r$.
- There exists $p \in I$ with $p \xrightarrow{uv} r$.

This shows $v \in L(\mathcal{B})$ if and only if $uv \in L(\mathcal{A})$ and thus $L(\mathcal{B}) = u^{-1}L(\mathcal{A})$. If $\mathcal{A} = (Q, \cdot, q_0, F)$ is deterministic, then the above constructions yields the automaton $\mathcal{B} = (Q, \cdot, q_0', F)$ with initial state $q_0' = q_0 \cdot u$.

Next we consider right residuals. Suppose $\mathcal{A} = (Q, \delta, I, F)$ is a nondeterministic automaton with $\delta \subseteq Q \times \Sigma \times Q$. In this case we define $\mathcal{B} = (Q, \delta, I, F')$ with

$$F' = \left\{ p \in Q \;\middle|\; \exists q \in F \colon p \xrightarrow{u} q \right\}.$$

By a similar reasoning as for left residuals we see that $L(\mathcal{B}) = L(\mathcal{A})u^{-1}$. In summary, the constructions for residuals are obtained by adapting initial and final states; moreover, if $\mathcal{A}$ is deterministic, then $\mathcal{B}$ is deterministic, too.

Quotients

The *left quotient* of a language $L \subseteq \Sigma^*$ by $K \subseteq \Sigma^*$ is

$$K^{-1}L = \{\, v \in \Sigma^* \mid uv \in L \text{ for some } u \in K \,\}.$$

Thus, quotients are generalizations of residuals. Right quotients LK^{-1} are defined symmetrically, that is $LK^{-1} = \{\, u \in \Sigma^* \mid uv \in L \text{ for some } v \in K \,\}$. We say that a language L' is a *quotient* of L if it is either a left or a right quotient of L. We have

$$K^{-1}L = \bigcup_{u \in K} u^{-1}L. \tag{4.1}$$

Every regular language has only finitely many left residuals (and symmetrically it has only a finite number of right residuals) since all residuals are accepted by the same automaton with varying initial states. Thus, if L is regular, then the union in Equation (4.1) is finite. This shows that for a regular language L and an arbitrary language K the quotient $K^{-1}L$ (and symmetrically LK^{-1}) is regular. This does not mean that we can always compute $K^{-1}L$ since the condition "$u \in K$" in Equation (4.1) might be undecidable. As an intermediate step between left residuals and arbitrary left quotients we consider the following problem.

Input: Finite Σ^*-automata $\mathcal{A}_1$ and $\mathcal{A}_2$.

Output: A finite Σ^*-automaton $\mathcal{B}$ with $L(\mathcal{B}) = L(\mathcal{A}_2)^{-1}L(\mathcal{A}_1)$.

Let $\mathcal{A}_i = (Q_i, \delta_i, I_i, F_i)$ with $\delta_i \subseteq Q_i \times \Sigma \times Q_i$. Consider the product automaton $\mathcal{A}_1 \times \mathcal{A}_2$. Using

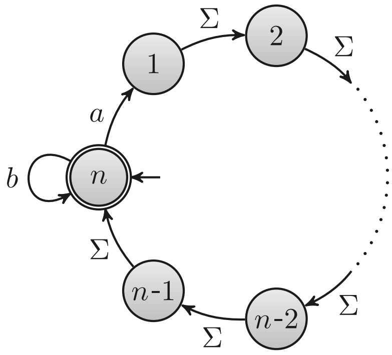
$$I' = \{\, q \in Q_1 \mid \exists f \in F_2 \colon (q, f) \text{ is reachable in } \mathcal{A}_1 \times \mathcal{A}_2 \,\}$$

as the initial states we obtain the automaton $\mathcal{B} = (Q_1, \delta_1, I', F_1)$, *i.e.*, $\mathcal{B}$ is obtained from $\mathcal{A}_1$ by modifying the initial states and the product automaton $\mathcal{A}_1 \times \mathcal{A}_2$ was only used for computing the new set of initial states. Let $r \in Q$ and $v \in \Sigma^*$. Then the following conditions are equivalent:

- There exists $q \in I'$ with $q \xrightarrow{v} r$ in $\mathcal{B}$.
- There exists $q \in Q_1$ with $q \xrightarrow{v} r$ in $\mathcal{A}_1$ and there exists $f \in F_2$ such that (q, f) is a reachable state of the product automaton $\mathcal{A}_1 \times \mathcal{A}_2$.
- There exist pairs of states $(q, f) \in Q_1 \times F_2$ and $(i_1, i_2) \in I_1 \times I_2$ and there exists a word $u \in \Sigma^*$ such that $(i_1, i_2) \xrightarrow{u} (q, f)$ in $\mathcal{A}_1 \times \mathcal{A}_2$ and $q \xrightarrow{v} r$ in $\mathcal{A}_1$.
- There exists $u \in L(\mathcal{A}_2)$ and there exists $i_1 \in I_1$ with $i_1 \xrightarrow{uv} q$.

This shows $L(\mathcal{B}) = L(\mathcal{A}_2)^{-1}L(\mathcal{A}_1)$. If $\mathcal{A}_1$ is deterministic, then $L(\mathcal{B})$ can be nondeterministic due to multiple initial states. Thus, for computing a deterministic automaton for $L(\mathcal{A}_2)^{-1}L(\mathcal{A}_1)$ one might need to apply the powerset construction which for an n-state automaton $\mathcal{A}_1$ yields a deterministic automaton for the quotient $L(\mathcal{A}_2)^{-1}L(\mathcal{A}_1)$ with at most $2^n - 1$ states (the empty set $\emptyset$ is no state of the powerset construction of $\mathcal{B}$). The following example shows that this bound is tight. It is a slight modification of an example of Yu, Zhuang and Salomaa [**?**].

Example 4.6. Let $\Sigma = \{a, b\}$ and $Q = \{1, \ldots, n\}$. We consider the following deterministic n-state Σ^*-automaton $\mathcal{A} = (Q, \cdot, n, \{n\})$ for the language $L = \left(b^* \cup a\Sigma^{n-1}\right)^*$.



A nondeterministic automaton for the left quotient $(\Sigma^*)^{-1}L$ of L by Σ^* is obtained from $\mathcal{A}$ by making all states initial, *i.e.*, we consider the nondeterministic automaton $\mathcal{B} = (Q, \delta, Q, \{n\})$ with transition relation $\delta = \{\, (i, u, j) \in Q \times \Sigma \times Q \mid i \cdot u = j \,\}$ such that every state of $\mathcal{B}$ is initial. We claim that the powerset construction $\mathcal{P}(\mathcal{B})$ of $\mathcal{B}$ has $2^n - 1$ states and that it is the minimal automaton of $(\Sigma^*)^{-1}L$. Consider a word $u = a_n \cdots a_1 \neq b^n$ with $a_i \in \Sigma$ and suppose $Q \cdot u = P$ in the powerset construction $\mathcal{P}(\mathcal{B})$. Then we have $j \in P$ if and only if $a_j = a$. Therefore, every nonempty subset of Q is a reachable state in $\mathcal{P}(\mathcal{B})$. Consider now two distinct states $P, P' \subseteq Q$ of $\mathcal{P}(\mathcal{B})$. Without loss of generality there exists $j \in P \setminus P'$. Then $n \in P \cdot a^{n-j}$ whereas $n \notin P' \cdot a^{n-j}$. This shows that $\mathcal{P}(\mathcal{B})$ is minimal. Its states are the nonempty subsets of Q. In particular, the minimal automaton of $(\Sigma^*)^{-1}L$ has $2^n - 1$ states.

We also note the interpretation of $\mathcal{B}$ as an automaton over $\{a, b, c\}^*$ yields a powerset construction with 2^n states since $\emptyset$ becomes a reachable state (for instance, using the word c of length 1). Therefore, the bound 2^n on the number of states in the powerset construction is tight. ◇

Computing automata for right quotients is easier in the sense that there is no exponential blow-up for deterministic automata. In the remainder of this section we consider the following problem:

Input: Finite Σ^*-automata $\mathcal{A}_1$ and $\mathcal{A}_2$.

Output: A finite Σ^*-automaton $\mathcal{B}$ with $L(\mathcal{B}) = L(\mathcal{A}_1)\, L(\mathcal{A}_2)^{-1}$.

Let $\mathcal{A}_i = (Q_i, \delta_i, I_i, F_i)$ with $\delta_i \subseteq Q_i \times \Sigma \times Q_i$. Consider the product automaton $\mathcal{A}_1 \times \mathcal{A}_2$ with states $Q_1 \times Q_2$ (even if not all states are reachable). We say that a state $(p_1, p_2) \in Q_1 \times Q_2$ is *co-reachable* if there exists $(f_1, f_2) \in F_1 \times F_2$ and $v \in \Sigma^*$ such that $(p_1, p_2) \xrightarrow{v} (f_1, f_2)$ in $\mathcal{A}_1 \times \mathcal{A}_2$. Using

$$F' = \{\, q \in Q_1 \mid \exists i_2 \in I_2 \colon (q, i_2) \text{ is co-reachable in } \mathcal{A}_1 \times \mathcal{A}_2 \,\}$$

as the final states we obtain the automaton $\mathcal{B} = (Q_1, \delta_1, I_1, F')$. As for left quotients one can verify that $L(\mathcal{B}) = L(\mathcal{A}_1)\, L(\mathcal{A}_2)^{-1}$. The main difference to left quotients is that $\mathcal{B}$ is deterministic whenever $\mathcal{A}_1$ is deterministic.

Other operations

We do not discuss algorithmic aspects of the operations *concatenation* and *Kleene star*. The reason is that the approach suggested by the Thompson construction (followed by removal of ε-transitions and possibly determinization) yields automata of asymptotically optimal size for these constructions, see e.g. [**?**].

4.4. Decision problems for automata

The word problem

For an automaton $\mathcal{A}$ over Σ^* the *word problem* is the following.

Input: A word $u = a_1 \cdots a_n$ with $a_i \in \Sigma$.
Question: Is $u \in L(\mathcal{A})$?

If $\mathcal{A}$ is deterministic, then we compute the state $p = q_0 \cdot u$ letter by letter and then we check whether or not $p \in F$. We have $p \in F$ if and only if $u \in L(\mathcal{A})$. In pseudocode, the algorithm is as follows.

Algorithm 1 The word problem for a deterministic automaton on input $a_1 \cdots a_n$

```
p ← q_0
for i ← 1, ..., n do p ← p · a_i
if p ∈ F then return true
else return false
```

The algorithm also works if Σ is the generating set of an arbitrary monoid M, and $\mathcal{A}$ is a deterministic M-automaton. The running time of the algorithm can be estimated by $\mathcal{O}(n)$ operations in the automaton $\mathcal{A}$.

If $\mathcal{A} = (Q, \delta, I, F)$ is a nondeterministic automaton with $\delta \subseteq Q \times \Sigma \times Q$, then we can implicitely use the powerset construction, *i.e.*, we set $P \cdot a = \{ q \in Q \mid \exists p \in P\colon (p, a, q) \in \delta \}$. The word problem for $\mathcal{A}$ is solved by computing $P = I \cdot u$, and then checking $P \cap F \neq \emptyset$. Thus the algorithm is as follows.

Algorithm 2 The word problem for a nondeterministic automaton on input $a_1 \cdots a_n$

```
P ← I
for i ← 1, ..., n do P ← P · a_i
if P ∩ F ≠ ∅ then return true
else return false
```

If $|Q| = m$, then this algorithm requires at most $\mathcal{O}(m^2 n)$ operations in the automaton $\mathcal{A}$. We note that the above algorithm for nondeterministic automata does not work for arbitrary monoids. There exist monoids M such that when given two sequences of generators $a_1 \cdots a_n$ and $b_1 \cdots b_m$ it is undecidable whether $a_1 \cdots a_n$ and $b_1 \cdots b_m$ represent the same element in M. Note that some factorization of an element might yield an accepting path whereas another factorization has no accepting path.

The emptiness problem

The *emptiness problem* is the following.

Input: A finite nondeterministic automaton $\mathcal{A}$.
Question: Is $L(\mathcal{A}) = \emptyset$?

For every initial state we check if there exists a path to some final state. This can be done in linear time, *e.g.*, by depth-first search. We note that non-emptiness is essentially graph reachability which is complete for nondeterministic logarithmic space (NL), a complexity class within deterministic polynomial time. The NL-algorithm for the Σ^*-automaton $\mathcal{A} = (Q, \delta, I, F)$ with $\delta \subseteq Q \times \Sigma \times Q$ can sketched as follows:

Algorithm 3 The emptiness problem for a nondeterministic automaton

```
nondeterministically guess some initial state $p \in I$
while $p \notin F$ do
    nondeterministically guess $a \in \Sigma$ and $q \in Q$ with $(p, a, q) \in \delta$
    $p \leftarrow q$
return "$L(\mathcal{A})$ is nonempty"
```

For a deterministic automaton $\mathcal{A} = (Q, \cdot, q_0, F)$, the algorithm can easily be adapted:

Algorithm 4 The emptiness problem for a deterministic automaton

```
$p \leftarrow q_0$
while $p \notin F$ do
    nondeterministically guess $a \in \Sigma$
    $p \leftarrow p \cdot a$
return "$L(\mathcal{A})$ is nonempty"
```

Since only the graph structure of the automaton matters, each of the algorithms can also be applied to automata over arbitrary monoids.

The intersection of a list of deterministic automata

When proving PSPACE lower bounds for decision problems on automata, the following theorem is very useful.

Theorem 4.1. *The following problem is* PSPACE*-complete.*

Input: *A list of deterministic* $\{a, b\}^*$*-automata* $\mathcal{A}_1, \ldots, \mathcal{A}_\ell$.

Question: *Is* $L(\mathcal{A}_1) \cap \cdots \cap L(\mathcal{A}_\ell) \neq \emptyset$*?*

Proof. For membership in PSPACE we consider the following algorithm. The algorithm works for automata $\mathcal{A}_i$ over some arbitrary alphabet Σ. Let q_0^i be the initial state of $\mathcal{A}_i$ and let F_i be its set of final states.

Algorithm 5 The intersection problem for a list of deterministic Σ^*-automata $\mathcal{A}_1, \ldots, \mathcal{A}_\ell$

```
$(p_1, \ldots, p_\ell) \leftarrow (q_0^1, \ldots, q_0^\ell)$
while $(p_1, \ldots, p_\ell) \notin F_1 \times \cdots \times F_\ell$ do
    nondeterministically guess $c \in \Sigma$
    $(p_1, \ldots, p_\ell) \leftarrow (p_1 \cdot c, \ldots, p_\ell \cdot c)$
return "$L(\mathcal{A}_1) \cap \cdots \cap L(\mathcal{A}_\ell) \neq \emptyset$"
```

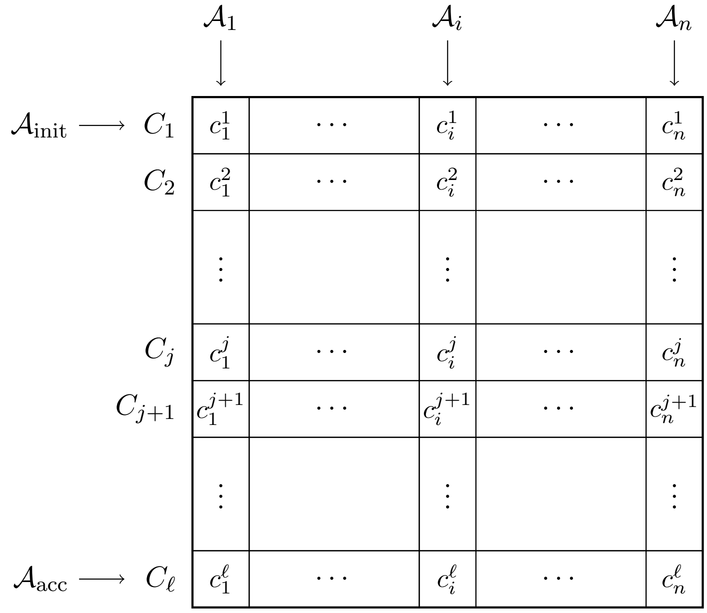
For showing PSPACE-hardness we use a reduction from the word problem for linearly bounded Turing machines, a PSPACE-complete problem. A linearly bounded Turing machine is a nondeterministic Turing machine which, on its tape, only uses the space of the input. Let M be a linearly space bounded nondeterministic Turing machine with input alphabet Σ and tape alphabet Γ. Let Q be the states of M and let $\delta \subseteq Q \times \Gamma \times Q \times \Gamma \times \{L, R\}$ be the transition relation. As usual, the letters L and R indicate the direction of the head. Suppose M has a unique initial state q_{init} and a unique accepting state q_{acc}.

Consider some input word $u \in \Sigma^*$ of length $n-1$. A *configuration* of M can be written as $c_1 \cdots c_n$ with $c_i \in \Gamma \cup Q$ such that $c_j \in Q$ for exactly one index j, meaning that M is in state c_j when reading symbol c_{j+1} at position j of the tape. Consider two configurations $C = c_1 \cdots c_n$ and $D = d_1 \cdots d_n$ with $C \vdash_M D$. Then the validity of the letter d_i only depends on $c_{i-1}c_ic_{i+1}$ and d_{i-1}, d_{i+1} (where c_{i-1}, c_{i+1}, d_{i-1}, or d_{i+1} might be empty), *i.e.*, for verifying $C \vdash_M D$ at position i it suffices to remember a fixed number of elements in $\Gamma \cup Q$.

$$\begin{array}{c|c|c|c|c|c}
C & \cdots & a & p & b & \cdots \\
D & \cdots & a & b' & q & \cdots
\end{array}
\qquad\qquad
\begin{array}{c|c|c|c|c|c}
C & \cdots & a & p & b & \cdots \\
D & \cdots & q & a & b' & \cdots
\end{array}$$

transition (p, b, q, b', R) transition (p, b, q, b', L)

By using some binary encoding of elements in $\Gamma \cup Q$ by words in $\{a, b\}^k$ for $k \in \mathcal{O}(\log(|\Sigma|+|Q|))$, we can write down configurations as elements of $\{a, b\}^m$ for $m = kn$. We consider computations as sequences of (encodings of) configurations $C_1, \ldots, C_\ell$ with $C_j \vdash_M C_{j+1}$ for all $j < \ell$. The integer ℓ is unknown. We use the concatenation $C_1 \cdots C_\ell \in \{a, b\}^*$ as the input for some automata. Let $\mathcal{A}_i$ be a deterministic automaton which checks consitency at position i, that is, the automaton $\mathcal{A}_i$ verifies $C_j \vdash_M C_{j+1}$ at position i for all $j \geq \ell$. This can be done with an automaton of polynomial size since it only has to remember some word in $\{a, b\}^{3k}$ from positions $i-1, i, i+1$ in C_j until reading these positions in C_{j+1}. This can be achieved by an automaton of polynomial size. Let $\mathcal{A}_{\text{init}}$ be a deterministic automaton for encodings of $q_{\text{init}}u\,(\Gamma \cup Q)^*$ and let $\mathcal{A}_{\text{acc}}$ be a deterministic automaton for encodings of $(\Gamma \cup Q)^*\,q_{\text{acc}}\,(\Gamma \cup Q)^*$; both automata have polynomial size.



C_1	c_1^1	$\cdots$	c_i^1	$\cdots$	c_n^1
C_2	c_1^2	$\cdots$	c_i^2	$\cdots$	c_n^2
	$\vdots$		$\vdots$		$\vdots$
C_j	c_1^j	$\cdots$	c_i^j	$\cdots$	c_n^j
C_{j+1}	c_1^{j+1}	$\cdots$	c_i^{j+1}	$\cdots$	c_n^{j+1}
	$\vdots$		$\vdots$		$\vdots$
C_ℓ	c_1^ℓ	$\cdots$	c_i^ℓ	$\cdots$	c_n^ℓ

The list of automata $\mathcal{A}_{\text{init}}, \mathcal{A}_1, \ldots, \mathcal{A}_n, \mathcal{A}_{\text{acc}}$ has the following property:

$$\begin{aligned}
u \in L(M) &\Leftrightarrow \text{there exists an accepting computation of } M \text{ on input } u \\
&\Leftrightarrow L(\mathcal{A}_{\text{init}}) \cap L(\mathcal{A}_1) \cap \cdots \cap L(\mathcal{A}_n) \cap L(\mathcal{A}_{\text{acc}}) \neq \emptyset.
\end{aligned}$$

Therefore, deciding the emptiness of the intersection of $\mathcal{A}_{\text{init}}, \mathcal{A}_1, \ldots, \mathcal{A}_n, \mathcal{A}_{\text{acc}}$ cannot be easier than deciding whether $u \in L(M)$. □

A naive approach to solving the problem in Theorem 4.1 might be the application of the product automaton construction, but this would yield an automaton which is exponential in the parameter ℓ.

The universality problem for deterministic automata

The *universality problem* for automata over Σ^* is defined as follows.

Input: A finite automaton $\mathcal{A}$ over Σ^*.

Question: Is $L(\mathcal{A}) = \Sigma^*$?

An automaton $\mathcal{A}$ with $L(\mathcal{A}) = \Sigma^*$ is called *universal.* If $\mathcal{A}$ is deterministic, then we can solve the emptiness problem for the complement automaton, *i.e.*, we check if there exists a path from the initial state to some non-final state. Therefore, the universality problem for deterministic automata can be solved in nondeterministic logarithmic space (*i.e.* in NL) as well as in deterministic linear time.

The universality problem for nondeterministic automata

Let $\mathcal{A} = (Q, \delta, I, F)$ with $\delta \subseteq Q \times \Sigma \times Q$ be a nondeterministic Σ^*-automaton. The non-universality problem on input $\mathcal{A}$ can be solved using the following nondeterministic polynomial space (PSPACE) algorithm:

Algorithm 6 The non-universality problem for a nondeterministic automaton

```
$P \leftarrow I$
while $P \cap F \neq \emptyset$ do
    nondeterministically guess $a \in \Sigma$
    $P \leftarrow P \cdot a$
return "$\mathcal{A}$ is not universal"
```

Remember that $P \cdot a = \{\, q \in Q \mid (p, a, q) \in \delta \text{ for some } p \in P \,\}$. This algorithm is nothing but the NL-algorithm for non-emptiness of $\overline{\mathcal{P}(\mathcal{A})}$, but polynomial space is required for writing down some state of the powerset construction. Note that nondeterministic PSPACE and deterministic PSPACE coincide by Savitch's Theorem, see *e.g.* [**?**]. Thus universality (and not just non-universality) of nondeterministic automata is in PSPACE, even if the alphabet Σ is part of the input. The following Theorem is an intermediate step towards PSPACE-completeness of the universality problem for nondeterministic automata.

Theorem 4.2. *The universality problem for finite nondeterministic automata over* $\{a, b\}^*$ *is* PSPACE-*complete.*

Proof. We have seen in Algorithm 6 that the problem is in PSPACE. For showing PSPACE-hardness we use a reduction from problem in Theorem 4.1. In fact we use the emptiness rather than non-emptiness, but this problem is also PSPACE-hard since PSPACE is closed under complemenation. Consider a list $\mathcal{A}_1, \ldots, \mathcal{A}_\ell$ of deterministic $\{a, b\}^*$-automata. Let $\mathcal{B} = \overline{\mathcal{A}_1} \cup \cdots \cup \overline{\mathcal{A}_\ell}$ be the union automaton of the respective complement automata. The automaton $\mathcal{B}$ has polynomial size and it satisfies:

$$\begin{aligned} & L(\mathcal{A}_1) \cap \cdots \cap L(\mathcal{A}_n) \cap L(\mathcal{A}_{\text{acc}}) = \emptyset \\ \Leftrightarrow\ & L(\overline{\mathcal{A}_1}) \cup \cdots \cup L(\overline{\mathcal{A}_\ell}) = \{a, b\}^* \\ \Leftrightarrow\ & L(\mathcal{B}) = \{a, b\}^* . \end{aligned}$$

Note that, due to the union automaton construction, $\mathcal{B}$ is nondeterministic even though the complement automata $\overline{\mathcal{A}_i}$ are deterministic. □

We note that the algorithm above and the hardness result in Theorem 4.2 immediately yields PSPACE-completeness of the universality problem for nondeterministic automata even if the alphabet is part of the input. This result is surprising since for deterministic automata the problem is in NL and it is known that NL is a proper subset of PSPACE (we have

NL $\subseteq$ P $\subseteq$ NP $\subseteq$ PSPACE, but to date only the inclusion NL $\subsetneq$ PSPACE is known to be strict). In particular, the universality problem for nondeterministic automata is *provably* more difficult than the universality problem for deterministic automata.

The inclusion problem

The *inclusion problem* for automata over Σ^* is defined as follows.

Input: Two finite automata $\mathcal{A}$ and $\mathcal{B}$ over Σ^*.

Question: Is $L(\mathcal{A}) \subseteq L(\mathcal{B})$?

Suppose $\mathcal{A}$ and $\mathcal{B}$ have m and n states, respectively. By using the deterministic single-state automaton for Σ^* as $\mathcal{A}$, we see that the inclusion problem cannot be easier than the universality problem. If $\mathcal{B}$ is deterministic, then one can check for emptiness of $L(\mathcal{A}) \cap L(\overline{\mathcal{B}})$ on the product automaton $\mathcal{A} \times \overline{\mathcal{B}}$. This can be done using $\mathcal{O}(mn)$ automaton operations. If $\mathcal{B}$ is nondeterministic, then the problem is PSPACE-hard by Theorem 4.2 and the above reduction. The following nondeterministic PSPACE-algorithm for non-inclusion shows that the problem is PSPACE-complete if $\mathcal{B}$ is allowed to be nondeterministic. Let $\mathcal{A} = (Q, \delta, I, F)$ and let $\mathcal{B} = (Q', \delta', I', F')$ such that all labels in δ and δ' are in Σ.

Algorithm 7 The non-inclusion problem for nondeterministic automata

```
(P, P') ← (I, I')
while $P \cap F = \emptyset$ or $P' \cap F' \neq \emptyset$ do
    nondeterministically guess $a \in \Sigma$
    $(P, P') \leftarrow (P \cdot a, P' \cdot a)$
return "$L(\mathcal{A}) \not\subseteq L(\mathcal{B})$"
```

Since PSPACE is closed under complement, this also yields an algorithm for inclusion (and not just "non-inclusion").

The equivalence problem

The *equivalence problem* for automata over Σ^* is the following.

Input: Two finite automata $\mathcal{A}$ and $\mathcal{B}$ over Σ^*.

Question: Is $L(\mathcal{A}) = L(\mathcal{B})$?

One can think of the equivalence problem as both directions $L(\mathcal{A}) \subseteq L(\mathcal{B})$ and $L(\mathcal{B}) \subseteq L(\mathcal{A})$. In particular, the equivalence problem is at most twice as difficult as the inclusion problem. Moreover, in the case that at least one of the automata is nondeterministic, Theorem 4.2 shows that equivalence is PSPACE-complete; for completeness, we explicitely give the PSPACE-algorithm for inequivalence on automata $\mathcal{A} = (Q, \delta, I, F)$ and $\mathcal{B} = (Q', \delta', I', F')$.

Algorithm 8 The inequivalence problem for nondeterministic automata

```
$(P, P') \leftarrow (I, I')$
while $P \cap F = \emptyset \Leftrightarrow P' \cap F' = \emptyset$ do
    nondeterministically guess $a \in \Sigma$
    $(P, P') \leftarrow (P \cdot a, P' \cdot a)$
return "$L(\mathcal{A}) \neq L(\mathcal{B})$"
```

Naturally, the same approach leads to an NL-algorithm for deterministic automata; and this algorithm can easily be implemented as a reachability problem on the product automaton $\mathcal{A} \times \mathcal{B}$. One just hast to answer the question, whether there exists a reachable state in $(F \times Q' \setminus F') \cup (Q \setminus F \times F')$. If $\mathcal{A}$ has m states and $\mathcal{B}$ has n states, then the running time would be $\mathcal{O}(mn)$, *i.e.* quadratic.

The Hopcroft-Karp equivalence test

In this section, we consider the "almost linear" equivalence test by Hopcroft and Karp for deterministic automata [**?**]. It relies on two operations on sets: *Union* and *Find*. The idea is that the algorithm starts with the partition where every element forms a singleton, and during the run of the algorithm partition classes are successively united. Then $\mathrm{Find}(x) = \mathrm{Find}(y)$ holds if and only if, in the current situation, x and y belong to the same class. The operation $\mathrm{Union}(x, y)$ unites the classes of x and y. Suppose $\mathcal{A} = (Q, \cdot, q_0, F)$ and $\mathcal{B} = (Q', \cdot, q'_0, F')$ are the given deterministic automata. Let $\tilde{Q} = Q \cup Q'$ be the disjoint union of the state sets. The algorithm starts with the partition where every element has its own class. The algorithm either returns that the automata are equivalent, or it computes a word u with either $u \in L(\mathcal{A}) \setminus L(\mathcal{B})$ or $u \in L(\mathcal{B}) \setminus L(\mathcal{A})$. Let $\mathcal{L}$ be a list of elements (p, p', u) with $p \in Q$, $p' \in Q'$ and $u \in \Sigma^*$.

Algorithm 9 Hopcroft-Karp equivalence test for deterministic automata

```
$\mathcal{L} \leftarrow \{(q_0, q'_0, \varepsilon)\}$
while $\mathcal{L} \neq \emptyset$ do
    Choose $(p, p', u) \in \mathcal{L}$ and remove this triple from $\mathcal{L}$
    if $\mathrm{Find}(p) \neq \mathrm{Find}(p')$ then
        if $(p, p') \in (F \times Q' \setminus F') \cup (Q \setminus F \times F')$ then
            return $u$
        else
            $\mathrm{Union}(p, p')$
            for all $a \in \Sigma$ do add $(p \cdot a,\ p' \cdot a,\ ua)$ to $\mathcal{L}$
return "$L(\mathcal{A}) = L(\mathcal{B})$"
```

Note that the algorithm executes $\mathrm{Union}(p, p')$ only for states with $\mathrm{Find}(p) \neq \mathrm{Find}(p')$. An important invariant is that all triples (p, p', u) which at some point occur in $\mathcal{L}$ satisfy $p = q_0 \cdot u$ and $p' = q'_0 \cdot u$. Hence, if the algorithm returns $u \in \Sigma^*$, then we have $u \in \big(L(\mathcal{A}) \setminus L(\mathcal{B})\big) \cup \big(L(\mathcal{B}) \setminus L(\mathcal{A})\big)$ and thus $L(\mathcal{A}) \neq L(\mathcal{B})$. For the correctness of the above algorithm it therefore remains to prove the following proposition.

Proposition 4.3. *If the Hopcroft-Karp equivalence test returns "$L(\mathcal{A}) = L(\mathcal{B})$", then the automata $\mathcal{A}$ and $\mathcal{B}$ are equivalent.*

Proof. We write $\mathrm{Find}_i(p) = \mathrm{Find}_i(p')$, if we have $\mathrm{Find}(p) = \mathrm{Find}(p')$ after the i-th iteration of the while-loop. Similarly, $\mathrm{Find}(p) = \mathrm{Find}(p')$ says that this query is true at the end of the algorithm.

Claim 1. *If* $\mathrm{Find}(p) = \mathrm{Find}(p')$, *then* $\mathrm{Find}(p \cdot a) = \mathrm{Find}(p' \cdot a)$ *for all* $a \in \Sigma$.

Proof of Claim 1: Suppose the contrary. Let i be minimal such that for some $(p, p', a) \in Q \times Q' \times \Sigma$ we have $\mathrm{Find}_i(p) = \mathrm{Find}_i(p')$ and $\mathrm{Find}(p \cdot a) \neq \mathrm{Find}(p' \cdot a)$. Note that $i > 0$ since otherwise $p = p'$ and this would contradict $\mathrm{Find}(p \cdot a) \neq \mathrm{Find}(p' \cdot a)$. Since $\mathrm{Find}_{i-1}(p) \neq \mathrm{Find}_{i-1}(p')$, we have $\mathrm{Find}_i(p) = \mathrm{Find}_i(p')$ due to some instruction $\mathrm{Union}(r, s)$ in the i-th iteration. We assume $\mathrm{Find}_{i-1}(r) = \mathrm{Find}_{i-1}(p)$ and $\mathrm{Find}_{i-1}(s) = \mathrm{Find}_{i-1}(p')$ since otherwise we may interchange the roles of r and s. After the operation $\mathrm{Union}(r, s)$ we add some element $(r \cdot a, s \cdot a, ua)$ to the list $\mathcal{L}$, and for all elements (q, q', u) on the list $\mathcal{L}$ we eventually have $\mathrm{Find}(q) = \mathrm{Find}(q')$. This shows $\mathrm{Find}(r \cdot a) = \mathrm{Find}(s \cdot a)$. By choice of i we have

$$\mathrm{Find}(p \cdot a) = \mathrm{Find}(r \cdot a) = \mathrm{Find}(s \cdot a) = \mathrm{Find}(p' \cdot a),$$

contradicting the assumption. This concludes the proof of Claim 1. □

As usual, let $L(q) = \{\, u \in \Sigma^* \mid q \cdot u \text{ is a final state} \,\}$. This allows us to describe an important property of the state partition computed by the algorithm.

Claim 2. *If* $\mathrm{Find}(p) = \mathrm{Find}(p')$*, then* $L(p) = L(p')$*.*

Proof of Claim 2: For all states p, p' with $\mathrm{Find}(p) = \mathrm{Find}(p')$ we show $u \in L(p) \Leftrightarrow u \in L(p')$ by induction on $|u|$. If $u = \varepsilon$, then this is true since we never unite final and non-final states. Let now $u = au'$ with $a \in \Sigma$ and consider the states $q = p \cdot a$ and $q' = p' \cdot a$. Claim 1 yields $\mathrm{Find}(q) = \mathrm{Find}(q')$, and by induction we see that $u' \in L(q) \Leftrightarrow u' \in L(q')$. Thus

$$au' \in L(p) \Leftrightarrow u' \in L(q) \Leftrightarrow u' \in L(q') \Leftrightarrow au' \in L(p').$$

This concludes the proof of Claim 2. ▫

Since (q_0, q_0', ε) is an element in the list $\mathcal{L}$, we eventually have $\mathrm{Find}(q_0) = \mathrm{Find}(q_0')$. By Claim 2 we obtain $L(\mathcal{A}) = L(q_0) = L(q_0') = L(\mathcal{B})$ as desired. □

The following proposition yields an upper bound on the number of Union and Find operations. Suppose $|Q| = m$ and $|Q'| = n$.

Proposition 4.4. *The algorithm executes at most*

(a) $m + n - 1$ Union *operations, and*

(b) $1 + |\Sigma| (m + n - 1)$ Find *comparisons.*

Moreover, if it returns a word $u \in \Sigma^*$ *after* k Union *operations, then* $|u| \leq k$*.*

Proof. Property (a) is trivial since we only unite disjoint subsets. Property (b): Every Find comparison is due to some triple in the list $\mathcal{L}$. After every Union operations, we add $|\Sigma|$ triples to $\mathcal{L}$. Together with the initial element (q_0, q_0', ε) in the list $\mathcal{L}$, this yields the desired bound. For the last statement, note that every Union operation increases the length of the longest word in the list $\mathcal{L}$ at most by 1. □

Proposition 4.4 immediately yields the following corollary.

Corollary 4.5. *The Hopcroft-Karp equivalence test requires at most* $\mathcal{O}(|\Sigma| (m + n))$ Union-Find *operations.* □

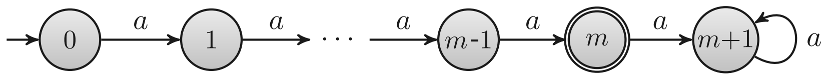
Another consequence of Proposition 4.4 is a linear bound on the length of a shortest witness for the inequivalence of automata. Moreover, this bound does not depend on the size of the alphabet.

Corollary 4.6. *Let* $\mathcal{A}$ *and* $\mathcal{B}$ *be deterministic* Σ^**-automata with* m *and* n *states, respectively. If* $L(\mathcal{A}) \neq L(\mathcal{B})$*, then there exists a word* $u \in \Sigma^*$ *of length at most* $m + n - 2$ *with* $u \in \big(L(\mathcal{A}) \setminus L(\mathcal{B})\big) \cup \big(L(\mathcal{B}) \setminus L(\mathcal{A})\big)$*.*

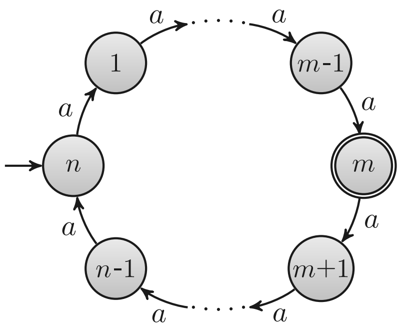
Proof. If $L(\mathcal{A}) \neq L(\mathcal{B})$, then by Proposition 4.3 the Hopcroft-Karp equivalence test returns a word $u \in \big(L(\mathcal{A}) \setminus L(\mathcal{B})\big) \cup \big(L(\mathcal{B}) \setminus L(\mathcal{A})\big)$. This can only happen before the $(m - n + 1)$-th Union operation (since after $m - n + 1$ Union operations, all states have the same Find-value). Proposition 4.4 yields the desired bound. □

The following example shows that the bound in Corollary 4.6 is tight.

Example 4.7. Let $m \leq n$ be two positive integers. First suppose $m < n$. Then we consider the following two deterministic a^*-automata $\mathcal{A}$ and $\mathcal{B}$ with $m + 2$ and n states, respectively. The automaton $\mathcal{A}$ is:

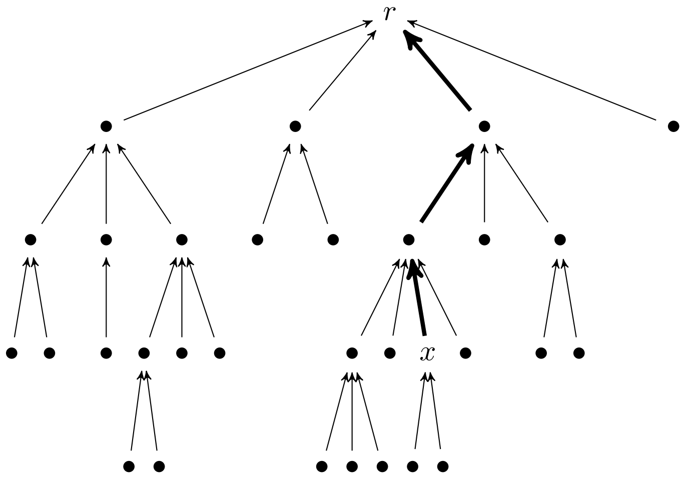


It accepts the singleton language $\{a^m\}$. The automaton $\mathcal{B}$ is given by:



We have $L(\mathcal{B}) = \left\{ a^{m+kn} \;\middle|\; k \geq 0 \right\}$. In particular $L(\mathcal{A}) \subsetneq L(\mathcal{B})$ and a^{m+n} is the shortest word in $L(\mathcal{B}) \setminus L(\mathcal{A})$. In the case $m = n$, we also choose the state 0 of the automaton $\mathcal{A}$ is final, *i.e.*, we then have $L(\mathcal{A}) = \{\varepsilon, a^n\}$ and $L(\mathcal{B}) = (a^n)^*$. ◇

In the remainder of this section, we briefly sketch one possible implementation of the operations Union and Find. Every class of the partition is represented by a tree in which the nodes have a pointer to their parents. Moreover, at every root we remember the number of nodes in the tree. The operation Find(x) returns the root of the tree containing x. All we have to do in order to compute Find(x) is to chase the pointers from nodes to their parent, starting with node x. For instance, in the following example, Find(x) would return r.



Let size(x) denote the number of nodes in the tree of x. Remember that we store size(x) at the node Find(x). Consider elements x, y with Find$(x) = r \neq s =$ Find(y) and suppose size$(x) \geq$ size(y). Then for the operation Union(x, y) we introduce a pointer from s to r, *i.e.*, we attach the smaller tree to the bigger one. In particular, after this operation we have Find$(y) = r$.

Let height(x) denote the length of a longest path in the tree of x. We claim that when starting with singleton sets for all elements and then executing successive Union operations, then at any time $2^{\mathrm{height}(x)} \leq \mathrm{size}(x)$. This is true at the beginning since singletons have height 0. Let now size$(x) \geq$ size(y) and consider the operation Union(x, y). Let height′ and size′ denote the respective values after the execution of this operation. If height′$(x) =$ height(x), then there is nothing to show. Let now height′$(x) >$ height(x). Then height′$(x) =$ height$(y) +$ 1. Since size$(x) \geq$ size(y), we have

$$2^{\mathrm{height}'(x)} = 2 \cdot 2^{\mathrm{height}(y)} \leq 2 \cdot \mathrm{size}(y) \leq \mathrm{size}(x) + \mathrm{size}(y) = \mathrm{size}'(x),$$

as desired.

Let n be the number of elements. The above estimate shows that (at any time) the running time of both operations Union(x, y) and Find(x) is in $\mathcal{O}(\log n)$.

There is a big improvement to the above algorithm which is called *path compression*. The idea is that after every execution of the procedure Find, we redirect the pointers of all nodes we see during this execution directly to the root. This is done by chasing the path a second time. The effect is that the subsequent Find operations for some elements having a predecessor on this path will be faster. An amortized analysis shows that this yields the inverse Ackermann function as an asymptotic running time for both Union and Find [**?**]. Since for all reasonable values, this function is less than 5, this coins the term "almost linear" in the running time of the Hopcroft-Karp equivalence test.

Identity testing

Identity testing addresses the question whether two given automata are identical, *i.e.*, can one automaton be obtained from the other by renaming the states. For two deterministic Σ^*-automata with n states each, this can be solved in time $\mathcal{O}(|\Sigma|\, n)$: We can assume that $\Sigma = \{a_1, \ldots, a_m\}$. Now, we number the states from 1 to n using a depth-first traversal; during this traversal, we always visit outgoing a_i-transition before considering a_{i+1}-transitions. Then for the two states p, q with number i in both automata, we have to check whether the number of $p \cdot a$ and of $q \cdot a$ is the same for all $a \in \Sigma$, and whether p is final if and only if q is final.

We note that this algorithm would not work for non-deterministic automata since for different outgoing a-transitions there is no uniform tie-breaker which one to process first in a depth-first traversal. If all transitions have the same label all states are both initial and final, then identity testing is nothing but graph isomorphism, a problem with an unresolved complexity to date.

The main application of identity testing for deterministic automata is the use of minimization algorithms for equivalence tests. This can be done as follows. Given two deterministic automata $\mathcal{A}$ and $\mathcal{B}$, we minimize both $\mathcal{A}$ and $\mathcal{B}$ and then we check whether the resulting automata are identical. The complexity of this procedure is dominated by the complexity of the minimization algorithm in use. In particular, minimizing automata cannot be faster than equivalence testing.

4.5. Minimization algorithms

The problem of minimization is the following. Given a finite deterministic automaton $\mathcal{A}$ we want to compute the minimal automaton accepting $L(\mathcal{A})$. As it turns out, many of the known algorithms can also be applied in the general setting of M-automata.

Moore's algorithm

Let M be an arbitrary monoid, let $\mathcal{A} = (Q, \cdot, q_0, F)$ be an M-automaton, and let $L = L(\mathcal{A})$. We can assume that all states are reachable, *i.e.*, we have $Q = q_0 \cdot M$. For a state $p \in Q$ we define $L(p) = \{\, v \in M \mid p \cdot v \in F \,\}$. For $p \in Q$ we set $[p] = \{\, q \in Q \mid L(p) = L(q) \,\}$. We define the automaton $\mathcal{B}_\mathcal{A}$ with states $\{\, [p] \mid p \in Q \,\}$. The initial state of $\mathcal{B}_\mathcal{A}$ is $[q_0]$ and the final states are $\{\, [f] \mid f \in F \,\}$. Note that $L(p) = L(q)$ implies $L(p \cdot u) = L(q \cdot u)$ for all $u \in M$. Therefore, the transition function $[p] \cdot u = [p \cdot u]$ is well-defined. Remember that the states of the minimal automaton $\mathcal{A}_L$ are the sets $L(u) = \{\, v \in M \mid uv \in L \,\}$. For every $u \in M$ we have

$$v \in L(q_0 \cdot u) \;\Leftrightarrow\; q_0 \cdot uv \in F \;\Leftrightarrow\; uv \in L \;\Leftrightarrow\; v \in L(u)$$

and thus $L(q_0 \cdot u) = L(u)$. In particular, $\mathcal{B}_\mathcal{A}$ is the the minimal automaton of L; the only difference is that we use different names for the states. In order to minimize $\mathcal{A}$, it remains to identify those states $p, q \in Q$ with $L(p) = L(q)$. This can be done with the following algorithm. Let M be generated by Σ.

(a) We start with the complete graph with vertices Q and edges $E = \{ \{p,q\} \mid p \neq q \in Q \}$.

(b) We mark all edges $\{p,q\} \in E$ with $p \in F$ and $q \notin F$.

(c) As long as there exist marked edges in E, we repeat the following procedure. We choose some marked edge $\{p',q'\} \in E$. Then we mark all unmarked edges $\{p,q\} \in E$ with $\{p',q'\} = \{p \cdot a, q \cdot a\}$ for some $a \in \Sigma$. Afterwards we remove the edge $\{p',q'\}$ from E.

Note that all marked edges are eventually removed.

Lemma 4.7. *We have $L(p) = L(q)$ if and only if $\{p,q\}$ is an edge in the remaining graph.*

Proof. $\Rightarrow$: We show that whenever an edge $\{p,q\}$ is marked, then we have $L(p) \neq L(q)$. If $\{p,q\}$ is marked in step (b), then $1 \in L(p) \setminus L(q)$. Let now $\{p,q\}$ be marked in step (c). Then there exists a previously marked edge $\{p',q'\}$ and a generator $a \in \Sigma$ with $\{p \cdot a, q \cdot a\} = \{p',q'\}$. By induction we have $L(p') \neq L(q')$. W.l.o.g. there exits $v \in L(p \cdot a) \setminus L(q \cdot a)$; otherwise we interchange the roles of p and q. It follows $av \in L(p) \setminus L(q)$ and $L(p) \neq L(q)$.

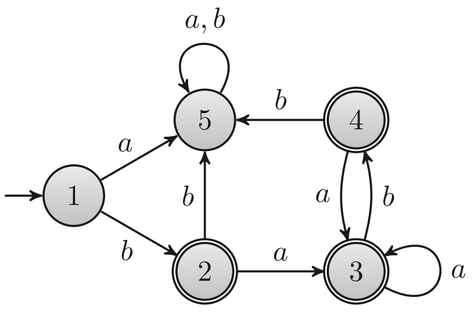
$\Leftarrow$: We say that an element $v \in M$ is a *witness*, if $v \in L(p) \setminus L(q)$ for some edge $\{p,q\}$ in the remaining graph. By contradiction, we assume that there exists a witness. Let $n \geq 0$ be minimal such that $v = a_1 \cdots a_n$ for some generators $a_i \in \Sigma$ is a witness. Then there exists an edge $\{p,q\}$ in the remaining graph with $v \in L(p) \setminus L(q)$. The case $n = 0$ is not possible since then $v = 1$ and $\{p,q\}$ would have been marked in step (b). Thus $n > 0$ and $a_2 \cdots a_n \in L(p \cdot a_1) \setminus L(q \cdot a_1)$. By minimality of n, the edge $\{p \cdot a_1, q \cdot a_1\}$ was removed. Before removal it was marked, and therefore in step (c) the edge $\{p,q\}$ was marked, too. Since marked edges are eventually removed, $\{p,q\}$ is not an edge in the remaining graph. This is a contradiction, showing that there exist no witnesses. This shows that every edge $\{p,q\}$ with $L(p) \neq L(q)$ is eventually removed. □

One can think of the above algorithm as a traversal of the following graph. The set of vertices is $\{ \{p,q\} \mid p \neq q \in Q \}$, and for every generator $a \in \Sigma$ and every vertex $\{p,q\}$ there exists an edge from $\{p \cdot a, q \cdot a\}$ to $\{p,q\}$. If Q and Σ are finite, then this graph has $\mathcal{O}(|Q|^2 \, |\Sigma|)$ edges. Therefore the running time of the above algorithm is $\mathcal{O}(|Q|^2 \, |\Sigma|)$.

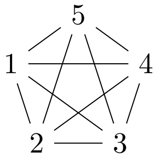
When implementing automata, one usually does not want to have sets $[p]$ as states. Instead, in every class $[p]$ one chooses some state as a representative. For any state $p \in Q$ let $\widehat{p}$ be the representative of $[p]$. The transition function is then given by $\widehat{p} \cdot a = \widehat{p \cdot a}$. With $\mathcal{O}(|Q| \, |\Sigma|)$, the running time of this step is negligible.

We note that we do not have to prove that the resulting automaton indeed is an M-automaton since it coincides with the minimal automaton of L which is already known to be an M-automaton.

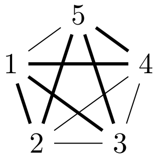
Example 4.8. In Example 3.1 we constructed the following deterministic $\{a,b\}^*$-automaton for the language $L = b\,\{ab, b\}^*$.



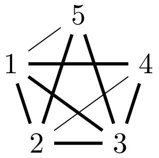
We apply Moore's minization algorithm to this automaton. The procedure starts with the complete graph on the set of states:



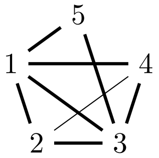
In the beginning we mark all edges between final and non-final states.



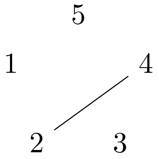
In the next step, we (arbitrarily) choose the marked edge $\{4,5\}$ which causes us to mark edges $\{2,3\}$ and $\{3,4\}$ since $(2 \cdot b, 3 \cdot b) = (5,4)$ and $(3 \cdot b, 4 \cdot b) = (4,5)$. Then we remove the edge $\{4,5\}$. This leads to the following graph:



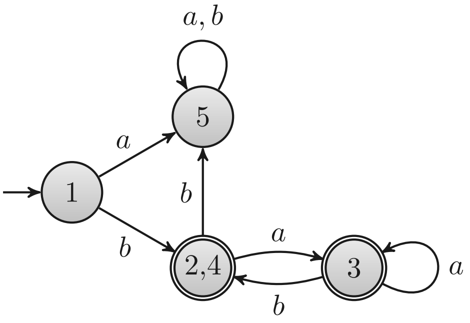
In the next iteration, we choose $\{2,5\}$ and we mark the edge $\{1,5\}$ since $(1 \cdot b, 5 \cdot b) = (2,5)$; then we remove the edge $\{2,5\}$.



In the remaining iterations, all marked edges are removed without causing additional marked edges.



Finally, the minimal automaton is obtained by combining the states 2 and 4 into a single state.



Note that only the "future" behavior of states counts for minimization (even though Moore's algorithm looks for predecessor edges). The states 2 and 4 have the same outgoing transitions which makes them identical in some rather obvious way; on the other hand the two states do not share the same incoming edges. ◇

Hopcroft's algorithm

We present Hopcroft's algorithm for minimizing a Σ^*-automaton $\mathcal{A} = (Q, \cdot, q_0, F)$ with n states. Even though the algorithm works for deterministic automata over arbitrary Σ-generated monoids, for simplicity we only present it for finitely generated free monoids. The running time of Hopcroft's minimization algorithm is in $\mathcal{O}(|\Sigma|\, n \log n)$; to date, no asymptotically faster algorithm is known. The algorithm computes the Myhill-Nerode equivalence relation $\equiv_{\mathcal{A}}$. Remember that reachable states $p, q \in Q$ satisfy $p \equiv_{\mathcal{A}} q$ if $L(p) = L(q)$, *i.e.*, if we have

$$p \cdot u \in F \Leftrightarrow q \cdot u \in F$$

for all $u \in \Sigma^*$. Let $[p] = \{\, q \in Q \mid p \equiv_{\mathcal{A}} q \,\}$ be the equivalence class of p. Representing $\equiv_{\mathcal{A}}$ as a subset of $Q \times Q$ is quadratic. Instead, it is more efficient to use the partition $\{\, [p] \mid p \in Q \,\}$ which can be stored in space $\mathcal{O}(n)$. In order to avoid unnecessary case distinctions, we frequently identify $\{\, \emptyset, P_1, \ldots, P_k \,\}$ with $\{\, P_1, \ldots, P_k \,\}$. For $\mathcal{P} = \{\, P_1, \ldots, P_k \,\}$ and $\mathcal{R} = \{\, R_1, \ldots, R_\ell \,\}$ with $P_i, R_j \subseteq Q$ we define

$$\mathcal{P} \wedge \mathcal{R} = \{\, P_i \cap R_j \mid 1 \leq i \leq k,\ 1 \leq j \leq \ell,\ P_i \cap R_j \neq \emptyset \,\}\,.$$

If $\mathcal{P}$ and $\mathcal{R}$ are partitions, then so is $\mathcal{P} \wedge \mathcal{R}$. A partition $\mathcal{R}$ *refines* $\mathcal{P}$ if for all $R \in \mathcal{R}$ there exists $P \in \mathcal{P}$ with $R \subseteq P$, *i.e.*, if $\mathcal{R} \wedge \mathcal{P} = \mathcal{R}$.

Lemma 4.8. *If $P = P_1 \cup P_2$ for $P \subseteq Q$, then $\{P_1, Q \setminus P_1\} \wedge \{P_2, Q \setminus P_2\}$ refines $\{P, Q \setminus P\}$.*

Proof. This immediately follows from $P_1, P_2 \subseteq P$ and $(Q \setminus P_1) \cap (Q \setminus P_2) = Q \setminus P$. □

A set S *splits* a set P if $\emptyset \neq P \cap S \neq P$. The *split* of P by S is (P_1, P_2) with $\{P_1, P_2\} = \{P \cap S, P \setminus S\}$ and $|P_1| \leq |P_2|$, *i.e.*, P_1 is the smaller one of the sets $P \cap S$ and $P \setminus S$; if both sets have the same size, then the choice is arbitrary. For $S \subseteq Q$ let $S \cdot a^{-1} = \{\, q \in Q \mid q \cdot a \in S \,\}$ be the states which have an outgoing a-transition to some state in S. For a partition $\mathcal{P}$ of Q, a subset $S \subseteq Q$ and a letter $a \in \Sigma$ we define

$$(S, a) \mid \mathcal{P} = \left\{ S \cdot a^{-1}, Q \setminus S \cdot a^{-1} \right\} \wedge \mathcal{P}.$$

Another view on $(S, a) \mid \mathcal{P}$ is as follows. The pair (S, a) defines the set $S \cdot a^{-1}$ which is used for splitting each class of $\mathcal{P}$. Note that $Q \setminus (S \cdot a^{-1}) = (Q \setminus S) \cdot a^{-1}$ and thus $Q \setminus S \cdot a^{-1}$ is well-defined. Let $\mathcal{A}$ be a Σ^*-automaton such that all states are reachable. Then Algorithm 10 is Hopcroft's minimization algorithm on input $\mathcal{A}$.

Algorithm 10 Hopcroft's algorithm for minimizing a Σ^*-automaton $\mathcal{A} = (Q, \cdot, q_0, F)$

```
 1: $\mathcal{P} \leftarrow \{F, Q \setminus F\}$
 2: $\mathcal{L} \leftarrow \{\, (F, a) \mid a \in \Sigma \,\}$
 3: while $\mathcal{L} \neq \emptyset$ do
 4:     Remove some pair $(S, a)$ from $\mathcal{L}$
 5:     for all $P \in \mathcal{P}$ which are split by $S \cdot a^{-1}$ do
 6:         Let $(P_1, P_2)$ be the split of $P$ by $S \cdot a^{-1}$          ▷ $P = P_1 \cup P_2$, $|P_1| \leq |P_2|$
 7:         $\mathcal{P} \leftarrow (\mathcal{P} \setminus \{P\}) \cup \{P_1, P_2\}$          ▷ Replace $P$ by $P_1, P_2$
 8:         for all $b \in \Sigma$ do
 9:             if $(P, b) \in \mathcal{L}$ then
10:                 $\mathcal{L} \leftarrow (\mathcal{L} \setminus \{(P, b)\}) \cup \{(P_1, b), (P_2, b)\}$     ▷ Replace $(P, b)$ by $(P_1, b), (P_2, b)$
11:             else
12:                 $\mathcal{L} \leftarrow \mathcal{L} \cup \{(P_1, b)\}$
13: return $\mathcal{P}$
```

$\mathcal{P} \leftarrow (S, a) \mid \mathcal{P}$; Update $\mathcal{L}$

As indicated, the instructions in lines 5 to 12 refine $\mathcal{P}$ by $(S, a) \mid \mathcal{P}$ and update the list $\mathcal{L}$. At some point, the current partition cannot be refined any further, and then the instruction in line 4 empties the list $\mathcal{L}$. Thus the while-loop terminates on all inputs. After every iteration of the while-loop one can consider the partition one obtains when using all pairs in $\mathcal{L}$ for refining $\mathcal{P}$. An important property of the resulting partitions is given by the following lemma.

Lemma 4.9. *Let $\mathcal{P}_i$ be the partition $\mathcal{P}$ after the i-th iteration of the while loop; in particular $\mathcal{P}_0 = \{F, Q \setminus F\}$. Let $\mathcal{L}_i$ be the list $\mathcal{L}$ after the i-th iteration, and let $\mathcal{R}_i = \bigwedge_{(S,a)\in\mathcal{L}_i}(S,a) \mid \mathcal{P}_i$. Then $\mathcal{R}_i$ refines $\mathcal{R}_{i-1}$ for all $i \geq 1$.*

Proof. The instruction in line 4 removes a pair (S,a) from $\mathcal{L}$, but then $\mathcal{P}$ is refined by this pair. The instruction in line 12 only increases the list $\mathcal{L}$. Therefore it suffices to show that updates of $\mathcal{L}$ using line 10 yield a refinement of the partition $\bigwedge_{(S,a)\in\mathcal{L}}(S,a) \mid \mathcal{P}$ for the current values of $\mathcal{L}$ and $\mathcal{P}$. Let $\mathcal{L}' = \{\, (S_1,a_1),\dots,(S_j,a_j),(P,b) \,\}$ be the list $\mathcal{L}$ before the execution of line 10 and let $\mathcal{L}'' = \{\, (S_1,a_1),\dots,(S_j,a_j),(P_1,b),(P_2,b) \,\}$ be the list after it. Let $\mathcal{R} = \bigwedge_{i=1}^{j}(S_i,a_i) \mid \mathcal{P}$, let $\mathcal{R}' = \bigwedge_{(S,a)\in\mathcal{L}'}(S,a) \mid \mathcal{P}$, and let $\mathcal{R}'' = \bigwedge_{(S,a)\in\mathcal{L}''}(S,a) \mid \mathcal{P}$. Then

$$\begin{aligned}
\mathcal{R}' &= \left\{P \cdot b^{-1},\, Q \setminus P \cdot b^{-1}\right\} \wedge \mathcal{R}, \\
\mathcal{R}'' &= \left\{P_1 \cdot b^{-1},\, Q \setminus P_1 \cdot b^{-1}\right\} \wedge \left\{P_2 \cdot b^{-1},\, Q \setminus P_2 \cdot b^{-1}\right\} \wedge \mathcal{R}.
\end{aligned}$$

The partition $\left\{P_1 \cdot b^{-1},\, Q \setminus P_1 \cdot b^{-1}\right\} \wedge \left\{P_2 \cdot b^{-1},\, Q \setminus P_2 \cdot b^{-1}\right\}$ refines $\left\{P \cdot b^{-1},\, Q \setminus P \cdot b^{-1}\right\}$ by Lemma 4.8 because $P = P_1 \cup P_2$ (and thus $P \cdot a^{-1} = P_1 \cdot a^{-1} \cup P_2 \cdot a^{-1}$). Therefore, $\mathcal{R}''$ refines $\mathcal{R}'$ as desired. □

Note that if the algorithm uses k iterations, then it returns the partition $\mathcal{R}_k$. Another view on this partition is as follows. Let (S_i,a_i) be the pair removed from $\mathcal{L}$ in line 4 in the i-th iteration of the while-loop and suppose the algorithm runs the while-loop k times. Hopcroft's algorithm then computes the partition

$$\left\{S_k \cdot a_k^{-1}, Q \setminus S_k \cdot a_k^{-1}\right\} \wedge \dots \wedge \left\{S_1 \cdot a_1^{-1}, Q \setminus S_1 \cdot a_1^{-1}\right\} \wedge \{F, Q \setminus F\}\,.$$

In particular, the order of the application of the pairs (S_i,a_i) does not matter for any fixed set of pairs $(S_1,a_1),\dots,(S_k,a_k)$. On the other hand, different choices for (S,a) in line 4 of the algorithm can yield different entries in the list $\mathcal{L}$ and therefore different intermediate partitions can occur. For instance, $\mathcal{L}$ could be implemented either as a stack or as a queue. The following proposition shows that, independent of the implementation, Hopcroft's algorithm always computes the partition defining the minimal automaton.

Proposition 4.10. *On input $\mathcal{A}$, Hopcroft's algorithm computes the Myhill-Nerode equivalence relation for $\mathcal{A}$.*

Proof. An important invariant of the algorithm is that before and after every iteration of the while-loop, all pairs $(S,a) \in \mathcal{L}$ satisfy $S \in \mathcal{P}$. For the correctness, we show that the partition computed by the algorithm represents the Myhill-Nerode equivalence relation $\equiv_{\mathcal{A}}$. We first show that $\equiv_{\mathcal{A}}$ refines the partition of the algorithm. Since $\equiv_{\mathcal{A}}$ refines the initial partition $\{F, Q \setminus F\}$, it suffices to prove the following claim: If $\equiv_{\mathcal{A}}$ refines $\mathcal{P}$, then $\equiv_{\mathcal{A}}$ also refines $(S,a) \mid \mathcal{P}$ for all $(S,a) \in \mathcal{P} \times \Sigma$. Let $p \equiv_{\mathcal{A}} q$. Then we have $p, q \in P$ for some $P \in \mathcal{P}$ and

$$p \in S \cdot a^{-1} \;\Leftrightarrow\; p \cdot a \in S \;\Leftrightarrow\; q \cdot a \in S \;\Leftrightarrow\; q \in S \cdot a^{-1},$$

where the second equivalence relies on $p \cdot a \equiv_{\mathcal{A}} q \cdot a$ and $S \in \mathcal{P}$. This shows that p and q are in the same class of the partition $(S,a) \mid \mathcal{P}$.

Next we show that Myhill-Nerode inequivalent states p, q are separated by the algorithm. Consider states p, q with $p \cdot u \in F \;\Leftrightarrow\; q \cdot u \notin F$ for some word $u \in \Sigma^*$. By induction on u we show that p and q end up in different sets of the final partition. For $u = \varepsilon$ this is true since the final partition refines the initial partition $\{F, Q \setminus F\}$. Let now $p \cdot a \not\equiv_{\mathcal{A}} q \cdot a$ for $a \in \Sigma$

such that $p \cdot a$, $q \cdot a$ are in different classes of the final partition. We show that p, q end up in different classes, too. Let $\mathcal{P}_i$ denote the partition $\mathcal{P}$ computed by the algorithm after the i-th iteration of the while-loop; for $i = 0$ we obtain the initial partition. Similarly, let $\mathcal{L}_i$ be the list $\mathcal{L}$ after the i-th iteration. After some iteration $i \geq 0$ of the while-loop we have:

- $p \cdot a \in P$ and $q \cdot a \in P'$ for disjoint classes $P, P' \in \mathcal{P}$, and
- $(P, a) \in \mathcal{L}$ or $(P', a) \in \mathcal{L}$.

This holds at the beginning (*i.e.* $i = 0$) or immediately after separating $p \cdot a$ from $q \cdot a$. Without loss of generality, let $(P, a) \in \mathcal{L}$. Since $p \in P \cdot a^{-1}$ and $q \notin P \cdot a^{-1}$, the states p, q are in different classes of the partition $(P, a) \mid \mathcal{P}$. Thus, by Lemma 4.9, the final partition separates p and q. □

Next, we analyze the running time of Hopcroft's algorithm. The presentation is based on the textbook by Beauquier, Berstel, and Chrétienne [**?**], see also [**?**]. We assume that the partition $\mathcal{P}$ can be implemented such that the following properties hold:

- Accessing the class of a state $q \in Q$ is in $\mathcal{O}(1)$.
- The size of a class $P \in \mathcal{P}$ can be determined in $\mathcal{O}(1)$.
- Enumerating the elements of a class $P \in \mathcal{P}$ is in $\mathcal{O}(|P|)$.
- Adding and removing an element to/from a class is in $\mathcal{O}(1)$.

We describe how to compute $\mathcal{P} \leftarrow (S, a) \mid \mathcal{P}$ in $\mathcal{O}(\left|S \cdot a^{-1}\right|)$. To this end, the classes of $\mathcal{P}$ in the for-loop in line 5 of Hopcroft's algorithm are processed simultaneously. First, for each $q \in S \cdot a^{-1}$ we do the following:

- We mark the class $P \in \mathcal{P}$ with $q \in P$.
- We increment a counter n_P for the class P.
- We add q to a list R_P.

Of course, we initially have $n_P = 0$ and $R_P = \emptyset$ for all marked classes P. In a second phase, we consider the marked classes P. If $n_P < |P|$, then for every state $q \in R_P$ we remove q from P (which yields $P \setminus R_P$ as a class), and we include the class R_P in the partition $\mathcal{P}$.

For the implementation of the list $\mathcal{L}$, we require that one can add and remove pairs in constant time. Moreover, one can check whether or not $(S, a) \in \mathcal{L}$ in constant time. Naturally, we represent the class S by a reference instead of copying the corresponding class in the partition.

Let $(S_1, a_1), \ldots, (S_k, a_k)$ be the pairs removed from $\mathcal{L}$ in line 4. Then the above implementation yields a running time of $\mathcal{O}(\sum_{i=1}^{k} \left|S_i \cdot a_i^{-1}\right|)$ for Hopcroft's algorithm. Thus the following proposition yields the desired bound of $\mathcal{O}(|\Sigma|\, n \log n)$.

Proposition 4.11. *We have* $\displaystyle\sum_{i=1}^{k} \left|S_i \cdot a_i^{-1}\right| \leq |\Sigma|\, n \log n$.

Proof. We fix some letter $a \in \Sigma$. We say that (S, a) is a *q-splitter* if $q \in S$. At any time, for every state q there is at most one q-splitter (S, a) in the list $\mathcal{L}$. When a q-splitter (S, a) is removed from $\mathcal{L}$ using the instruction in line 4 and later (not necessarily during the same iteration) a q-splitter (P_1, a) is added in line 12, then $2\,|P_1| \leq |S|$: In the setting of the algorithm, we have $P \subseteq S$ since P is currently the class of $\mathcal{P}$ which contains q and S has previously been a class of $\mathcal{P}$ containing q; thus $2\,|P_1| \leq |P_1| + |P_2| = |P| \leq |S|$. We conclude that at most $\log n$ pairs (S_i, a_i) form a q-splitter with $a_i = a$. Note that the size of a q-splitter could decrease during its stay in the list $\mathcal{L}$ by the instruction in line 10, but this does not affect the argument. For every state $p \in Q$ we have

$$p \in S \cdot a^{-1} \Leftrightarrow p \cdot a \in S \Leftrightarrow (S, a) \text{ is a } p \cdot a\text{-splitter.}$$

There are at most $\log n$ many pairs (S_i, a_i) which form a $p \cdot a$-splitter with $a_i = a$. By varying the letter a, we see that every state p is considered in at most $|\Sigma| \log n$ sets $S_i \cdot a_i^{-1}$. Taking the sum over all n states yields the desired bound. □

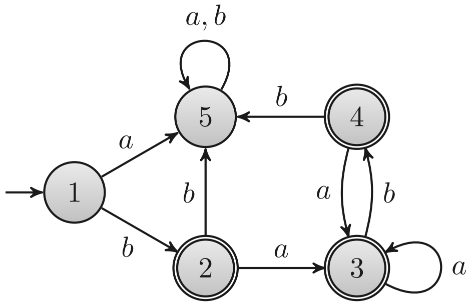
A slight optimization of Hopcroft's algorithm is to replace the initialization of the list $\mathcal{L}$ in line 2 by the following piece of code:

```
if $|F| \leq |Q \setminus F|$ then
    $\mathcal{L} \leftarrow \{ (F, a) \mid a \in \Sigma \}$
else
    $\mathcal{L} \leftarrow \{ (Q \setminus F, a) \mid a \in \Sigma \}$
```

That is, we use the class $Q \setminus F$ instead of F whenever there are more final than non-final states. This neither affects the correctness of the algorithm nor its asymptotic running time, but it slightly improves the constants hidden in the notation $\mathcal{O}(|\Sigma|\, n \log n)$.

There are several ways of implementing the list $\mathcal{L}$ and the partition $\mathcal{P}$ with the desired complexities, see *e.g.* [**?**]. For instance one could use a doubly linked list for representing a class $P \in \mathcal{P}$ together with a counter for $|P|$. Every state has a reference to the position within its class. The list $\mathcal{L}$ could also be organized as a doubly linked list. Every partition class P has an array of up to $|\Sigma|$ references to elements of $\mathcal{L}$, the entry for (P, a) is either null or it points to its position in $\mathcal{L}$.

Example 4.9. We revisit the deterministic $\{a, b\}^*$-automaton $\mathcal{A}$ for $L = b\{ab, b\}^*$ from Example 3.1.



Hopcroft's minimization algorithm on $\mathcal{A}$ starts with the partition $\mathcal{P} = \{ \{1, 5\}, \{2, 3, 4\} \}$. As mentioned above, in this case it is slightly better to initialize $\mathcal{L}$ using the non-final states, *i.e.*, at the beginning we have $\mathcal{L} = \{ (\{1, 5\}, a), (\{1, 5\}, b) \}$. Then a possible run of the algorithm could be:

1. In the first iteration we remove the pair $(\{1, 5\}, a)$ from $\mathcal{L}$. Note that $\{1, 5\} \cdot a^{-1} = \{1, 5\} \in \mathcal{P}$. Thus this does not affect the partition $\mathcal{P}$. The list is now $\mathcal{L} = \{ (\{1, 5\}, b) \}$.
2. In the second iteration we remove the pair $(\{1, 5\}, b)$ from $\mathcal{L}$. We have $\{1, 5\} \cdot b^{-1} = \{2, 4, 5\}$. This leads to the partition $\mathcal{P} = \{ \{1\}, \{5\}, \{2, 4\}, \{3\} \}$ and to the list $\mathcal{L} = \{ (\{1\}, a), (\{1\}, b), (\{3\}, a), (\{3\}, b) \}$ after the second iteration.
3. The remaining iterations do not affect $\mathcal{P}$.

Thus, not surprisingly, Hopcroft's algorithm computes the same minimal automaton as Moore's algorithm in Example 4.8. ◇

Remark 4.1. One can consider Hopcroft's algorithm as a particular way of implementing Moore's algorithm. In this setting, Hopcroft's algorithm combines two advantages. First, by using partitions it relies on a highly compressed representation of the graph used in Moore's algorithm. And second, when removing an element from a class P it deletes many edges from the graph in only one step. ◇

Brzozowski's algorithm

While Moore's algorithm and Hopcroft's algorithm work for deterministic automata over arbitrary monoids, the following algorithm due to Brzozowski only works for automata over free monoids Σ^*. One advantage of Brzozowski's algorithm is that it can also be applied if the input automaton is nondeterministic; as a drawback, it can construct intermediate automata of exponential size even if the input is deterministic.

The *reversal* of a word $u = a_1 \cdots a_n$ with $a_i \in \Sigma$ is $u^\rho = a_n \cdots a_1$. The reversal is obtained by reading the word from right to left. An obvious but crucial property of the reversal is $(u^\rho)^\rho = u$ for all $u \in \Sigma^*$, *i.e.*, reveral is an *involution*. The reversal of a language $L \subseteq \Sigma^*$ is $L^\rho = \{\, u^\rho \mid u \in L \,\}$. Let $\mathcal{A} = (Q, \delta, I, F)$ be a nondeterministic Σ^*-automaton with $\delta \subseteq Q \times \Sigma \times Q$. The reversal of $\mathcal{A}$ is $\mathcal{A}^\rho = (Q, \delta^\rho, F, I)$ with $\delta^\rho = \{\, (q, a, p) \mid (p, a, q) \in \delta \,\}$, *i.e.*, we reverse all transitions and we interchange initial and final states. Now, a word $u \in \Sigma^*$ has an accepting run in $\mathcal{A}$ if and only if u^ρ has an accepting run in $\mathcal{A}^\rho$. Thus $L(\mathcal{A}^\rho) = L(\mathcal{A})^\rho$. If $\mathcal{A}$ is deterministic, then $\mathcal{A}^\rho$ might be nondeterministic.

Theorem 4.12. *Let $\mathcal{B} = (Q, \cdot, q_0, F)$ be a deterministic Σ^*-automaton such that all states are reachable. Then $\mathcal{P}(\mathcal{B}^\rho)$ is the minimal Σ^*-automaton of $L(\mathcal{B})^\rho$.*

Proof. Remember that the powerset construction $\mathcal{P}(\mathcal{B}^\rho)$ only contains the reachable subsets of Q, the initial state is F and $P \subseteq Q$ is final if $q_0 \in P$. We have $L(\mathcal{P}(\mathcal{B}^\rho)) = L(\mathcal{B}^\rho) = L(\mathcal{B})^\rho$. Thus it remains to show that $\mathcal{P}(\mathcal{B}^\rho)$ is minimal.

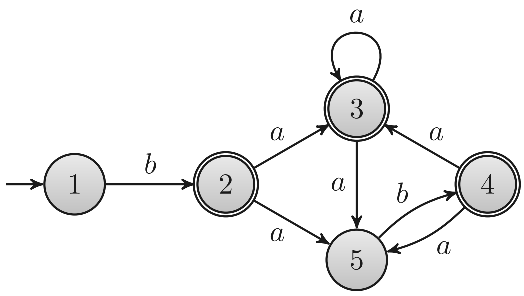
Claim. For any state $P \subseteq Q$ of $\mathcal{P}(\mathcal{B}^\rho)$ we have $q_0 \cdot u \in P$ if and only if $u^\rho \in L(P)$.

Proof of the claim: Let $q = q_0 \cdot u$ in $\mathcal{B}$ and let $P' = P \cdot u^\rho$ in $\mathcal{P}(\mathcal{B}^\rho)$. We have $q \xrightarrow{u^\rho} q_0$ in $\mathcal{B}^\rho$. Therefore, $q \in P$ imlies $q_0 \in P'$ and thus $u^\rho \in L(P)$. Conversely, if $q_0 \in P'$, then there exists a state $q' \in P$ with $q' \xrightarrow{u^\rho} q_0$. Since $\mathcal{B}$ is deterministic, we have $q' = q_0 \cdot u = q$; in particular, $q \in P$. This concludes the proof of the claim. ▫

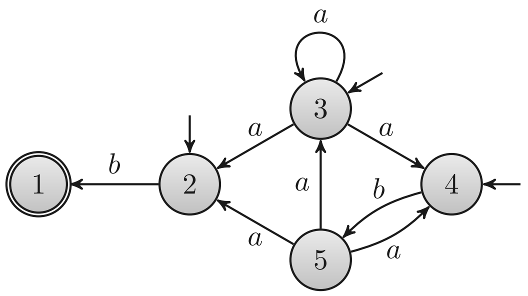
Consider two different states $P, P' \subseteq Q$. Let $q \in P \setminus P'$. Since all states in $\mathcal{B}$ are reachable, there exists a word $u \in \Sigma^*$ with $q_0 \cdot u = q$ in $\mathcal{B}$. By the above claim we have $u^\rho \in L(P) \setminus L(P')$. □

Let $\mathcal{A} = (Q, \delta, I, F)$ be a nondeterministic Σ^*-automaton with $\delta \subseteq Q \times \Sigma \times Q$. If we apply Theorem 4.12 with $\mathcal{B} = \mathcal{P}(\mathcal{A}^\rho)$, we see that $\mathcal{P}\big(\mathcal{P}(\mathcal{A}^\rho)^\rho\big)$ is the minimal automaton of $L(\mathcal{A})$. This means that "reverse-determinize-reverse-determinize" yields the minimal automaton. Even though there are two powerset constructions involved, the intermediate automata have at most exponential size since the minimal automaton of a nondeterministic automaton is at most exponential.

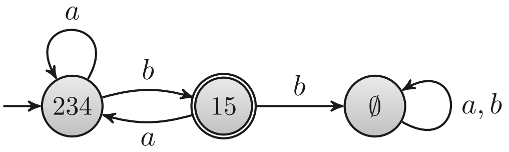
Example 4.10. We apply Brzozowski's minimization algorithm to the automaton constructed in Example 1.9. Let $\mathcal{A}$ be the following $\{a, b\}^*$-automaton for the language $L = b\,\{ab, a\}^*$.



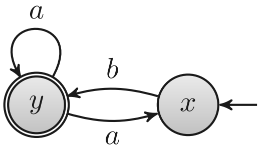
Its reversal $\mathcal{A}^\rho$ is:



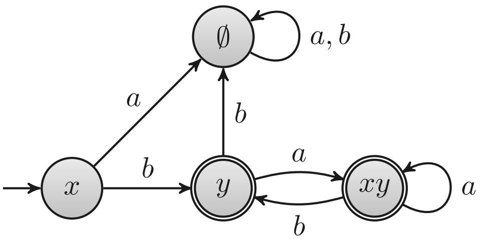
The powerset construction $\mathcal{P}(\mathcal{A}^\rho)$ yields:



After omitting the unreachable state $\emptyset$ in $\mathcal{P}(\mathcal{A}^\rho)^\rho$, we obtain the following nondeterministic automaton for L:



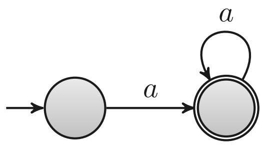
For better readability we used the name x for the state 15 and y for the state 234. The powerset constructing of $\mathcal{P}(\mathcal{A}^\rho)^\rho$ produces the following automaton:

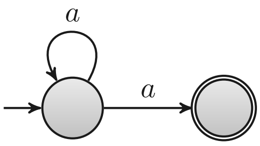


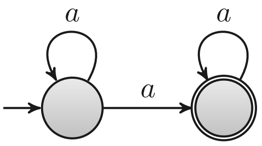
This is the minimal deterministic automaton of L. It is identical to the automaton constructed in Example 4.8. ◇

State reduction in nondeterministic automata

In general, there is no unique nondeterministic automaton with a minimal number of states. For instance, the following three a^*-automata for $L = aa^*$ are alle different (non-isomorphic):







There is no one-state automaton for L. Therefore, both automata have the minimal possible number of states. On the other hand non-deterministic automata can be much smaller than the minimal deterministic automaton, see *e.g.* Examples 3.2 and 4.6. Therefore, one might be interested in computing some nondeterministic automaton of minimal size. The following result shows that one cannot hope for an efficient algorithm for minimizing nondeterministic automata.

Theorem 4.13. *The following problem is* PSPACE*-complete.*

Input: *A finite nondeterministic automaton $\mathcal{A}$ over $\{a,b\}^*$ and an integer $k \geq 1$.*

Question: *Is there an equivalent nondeterministic automaton with at most k states.*

Proof. We show that the problem is in PSPACE for arbitrary alphabets Σ. Let $\mathcal{A}$ be a nondeterministic Σ^*-automaton and let $k \geq 1$. Then a PSPACE-algorithm can guess a nondeterministic Σ^*-auotomaton $\mathcal{B}$ and then check whether $L(\mathcal{A}) = L(\mathcal{B})$ using Algorithm 8.

For PSPACE-hardness, we use a reduction from universality of nondeterministic automata, see Theorem 4.2. Let $\mathcal{A}$ be a nondeterministic $\{a,b\}^*$-automaton. We use the followin procedure for deciding whether $\mathcal{A}$ is universal. First, we check that $a, b \in L(\mathcal{A})$. If this is the case, then we let $\mathcal{A}' = \mathcal{A}$, otherwise we let $\mathcal{A}'$ be some fixed automaton such that any automaton for $L(\mathcal{A}')$ requires at least two states. Now, $\mathcal{A}$ is universal if and only if there exists an equivalent 1-state automaton for $\mathcal{A}'$. This is because there is only one 1-state automaton $\mathcal{B}$ with $L(\mathcal{B}) \neq \emptyset$ having both an a- and a b-transition, and this automaton satisfies $L(\mathcal{B}) = \{a,b\}^*$. □

Next, we describe an efficient heuristic for reducing the number of states in a nondeterministic automaton. Let $\mathcal{A} = (Q, \delta, I, F)$ be a nondeterministic automaton with $\delta \subseteq Q \times \Sigma \times Q$. Then as in the case of deterministic automata, we can define $L(p)$ as the language accepted by the automaton (Q, δ, p, F). For a partition $\mathcal{P}$ of Q we can define the *quotient automaton* $\mathcal{A}/\mathcal{P} = (\mathcal{P}, \delta', I', F')$ with

$$\begin{aligned} \delta' &= \left\{ (P, a, P') \in \mathcal{P} \times \Sigma \times \mathcal{P} \,\middle|\, (p, a, p') \in \delta \text{ for } p \in P,\, p' \in P' \right\}, \\ I' &= \{ P \in \mathcal{P} \mid P \cap I \neq \emptyset \}, \\ F' &= \{ P \in \mathcal{P} \mid P \cap F \neq \emptyset \}. \end{aligned}$$

In general, we have $L(\mathcal{A}) \neq L(\mathcal{A}/\mathcal{P})$; for instance, let $L(\mathcal{A})$ be non-trivial and let $\mathcal{P} = \{Q\}$. We say that a partition $\mathcal{P}$ is *consistent* if for all classes $P \in \mathcal{P}$ and all states $p, q \in P$ we have $L(p) = L(q)$. For consistent partitions $\mathcal{P}$ we have $L(\mathcal{A}) = L(\mathcal{A}/\mathcal{P})$. A partition $\mathcal{P}$ is *stable* if the following two properties hold:

(a) Every class $P \in \mathcal{P}$ either only contains final states or it only contains non-final states.

(b) Whenever $p \xrightarrow{a} r$ is a transition in $\mathcal{A}$ and p, p' are in the same class, then there exists a state r' in the class of r such that $p' \xrightarrow{a} r'$ is a transtion.

Lemma 4.14. *Every stable partition is consistent.*

Proof. Let $\mathcal{P}$ be a stable partition, let $P \in \mathcal{P}$ be a class, let $p, p' \in P$, and let $u \in L(p)$. We want to show $u \in L(p')$. The proof is by induction on $|u|$. If $u = \varepsilon$, then p is a final state of $\mathcal{A}$. Thus, by property (a) in the definition of stability, p' is a final state, too. In particular, $u = \varepsilon$ is in $L(p')$.

Let now $u = au'$ for $a \in \Sigma$. Let $p \xrightarrow{a} r \xrightarrow{u'} q$ with $q \in F$ be a run on u. By property (b) of stability, there exists a state r' in the class of r such that $p' \xrightarrow{a} r'$. Induction yields $u' \in L(r')$. Thus $u = au' \in L(p')$ as desired. □

Remark 4.2. Let $\mathcal{A} = (Q, \cdot, q_0, F)$ be a deterministic Σ^*-automaton. An equivalence relation $\equiv$ on Q is a *congruence* if for all $p, q \in Q$ and all letters $a \in \Sigma$ the following two implications hold:

$$\begin{aligned} p \equiv q \;&\Rightarrow\; (p \in F \Leftrightarrow q \in F), \\ p \equiv q \;&\Rightarrow\; p \cdot a \equiv q \cdot a. \end{aligned}$$

A partition of Q is stable if and only if it induces a congruence. For instance, both the identity relation and the Myhill-Nerode equivalence form congruences. Moreover, the Myhill-Nerode congruence is the coarsest congruence, *i.e.*, all other congruences on Q refine the Myhill-Nerode congruence. ◇

For $S \subseteq Q$ and $a \in \Sigma$ let $S \cdot a^{-1} = \{\, p \in Q \mid \delta(p,a) \in S \,\}$ be the states which have an a-transition to S. For a partition $\mathcal{P}$, a subset $S \subseteq Q$ and a letter $a \in \Sigma$ we define $(S,a) \mid \mathcal{P} = \big\{S \cdot a^{-1}, Q \setminus \big(S \cdot a^{-1}\big)\big\} \wedge \mathcal{P}$ as the coarsest partition which refines both $\big\{S \cdot a^{-1}, Q \setminus \big(S \cdot a^{-1}\big)\big\}$ and $\mathcal{P}$. If $\mathcal{P} \neq (S,a) \mid \mathcal{P}$, then we say that (S,a) *splits* $\mathcal{P}$. A partition $\mathcal{P}$ of Q is stable if and only if the following two properties hold:

(a) $\mathcal{P}$ refines $\{F, Q \setminus F\}$.

(b) For all $S \in \mathcal{P}$ and all $a \in \Sigma$, the pair (S,a) does not split $\mathcal{P}$.

This immediately leads to the following algorithm for computing a stable partition:

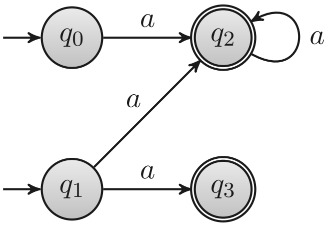
(a) Start with the partition $\mathcal{P} = \{F, Q \setminus F\}$.

(b) As long as there exists $S \in \mathcal{P}$ and $a \in \Sigma$ such that (S,a) splits $\mathcal{P}$, replace $\mathcal{P}$ by $(S,a) \mid \mathcal{P}$.

By Lemma 4.14, the resulting partition $\mathcal{P}$ of the algorithm satisfies $L(\mathcal{A}) = L(\mathcal{A}/\mathcal{P})$. We note that one cannot apply Hopcroft's algorithm in this setting since in general we have $Q \setminus \big(S \cdot a^{-1}\big) \neq (Q \setminus S) \cdot a^{-1}$. On the other hand, if $\mathcal{A}$ is deterministic, then the above algorithm computes the minimal automaton of $L(\mathcal{A})$. The two main differences in the setting of nondeterministic automata are:

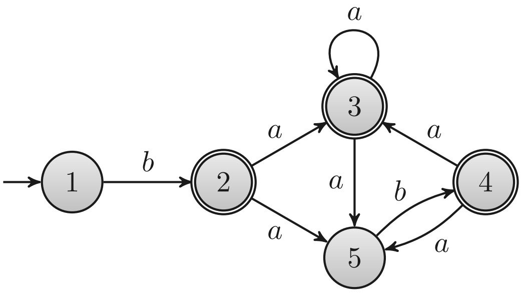
- We do not find all pairs of states (p,q) with $L(p) = L(q)$.
- We could combine two states p and q even if $L(p) \neq L(q)$.

Example 4.11. Consider the following nondeterministic automaton for aa^*:



The above algorithm would compute the identity relation, even though the non-stable partition $\{F, Q \setminus F\}$ would yield an equivalent smaller automaton. The algorithm splits q_0 and q_1 even though we have $L(q_0) = L(q_1)$, and we could combine q_2 and q_3 despite of $L(q_2) \neq L(q_3)$. For $S = \{q_3\}$ we have $S \cdot a^{-1} = \{q_1\}$ and $(Q \setminus S) \cdot a^{-1} = \{q_0, q_1, q_2\}$; in particular $Q \setminus \big(S \cdot a^{-1}\big) \neq (Q \setminus S) \cdot a^{-1}$. Also note that the above algorithm computes a nondeterministic automaton which is larger than the minimal automaton of aa^*. ◇

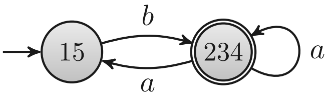
Example 4.12. We apply the above state reduction algorithm to the automaton constructed in Example 1.9. Let $\mathcal{A}$ be the following $\{a,b\}^*$-automaton for the language $L = b\,\{ab, a\}^*$.



The algorithm starts with the partition $\mathcal{P} = \{\,\{1,5\}, \{2,3,4\}\,\}$. We have

$$\begin{aligned}
\{1,5\} \cdot a^{-1} &= \{2,3,4\} \\
\{1,5\} \cdot b^{-1} &= \emptyset \\
\{2,3,4\} \cdot a^{-1} &= \{2,3,4\} \\
\{1,5\} \cdot b^{-1} &= \{1,5\}
\end{aligned}$$

and therefore, $\mathcal{P}$ is stable. The resulting automaton is:



In this case, the algorithm actually computes an equivalent nondeterministic automaton of minimal size. It is smaller than the corresponding minimal (deterministic) automaton. ◇