Regular languages[*]

We review the theory of finite automata and regular languages. The reason is treefold. Firstly, before studying Turing machines it is helpful to study a simpler type of computation. Secondly, automata model an important type of computation: computations *that require a constant amount of space*. Thirdly, throughout the course, many definitions will be illustrated using regular languages or regular expressions. In this section we review the basic theory needed to understand these examples. We follow Chapter 1 in the book of Michael Sipser: Introduction to the Theory of Computation.

Definitions. For a nonempty set Σ, the set of *strings* on Σ, denoted by Σ^*, is the set of all finite sequences of elements of Σ. The empty string, is denoted by ε. For example, $\{0,1\}^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, \dots\}$ represents the set of all bitstrings. An *alphabet* is a nonempty finite set, and the elements of an alphabet are called *symbols*. A *language* over an alphabet Σ is a subset of Σ^*.

1 Deterministic finite automata

A deterministic finite automaton is a device that takes a string over some alphabet as input and outputs either `accept` or `reject`. An automaton consists of a finite set of states that are connected by arrows, as illustrated in figure 1. From each state and for each symbol in the alphabet, there leaves precisely one arrow labelled by the symbol. One of the states is the *start state*, and there is a subset of states are called *accept states*. They are represented by double lines.

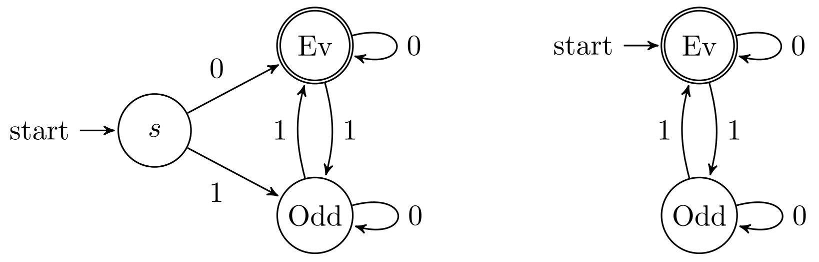


Figure 1: Schematic representation of automata.

On input a string w, the machine computes by moving a "finger" from one state to another. Initially, the finger is placed in the *start state*. For each subsequent symbol of w, the finger is moved along the arrow that is labelled by the symbol. After the last symbol is processed, the machine returns `accept` if the finger is located on an accept state; otherwise, it returns

[*]Bruno Bauwens, National Research University Higher School of Economics.
First version January 21, 2018.

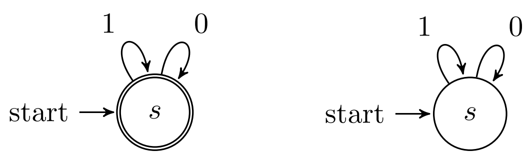


Figure 2: Left: an automata that accept all bitstrings (left) and no strings (right).

`reject`. For example, the automaton at the right of figure 1 has two states called *Ev* and *Odd*. On input 010010 it will visit them as follows:

$$\text{Ev} \xrightarrow{0} \text{Ev} \xrightarrow{1} \text{Odd} \xrightarrow{0} \text{Odd} \xrightarrow{0} \text{Odd} \xrightarrow{1} \text{Ev} \xrightarrow{0} \text{Ev},$$

and accept the string, because Ev is an accept state. More generally, this automaton accepts all strings with an even number of 1s and rejects all other strings. At the left, an automaton that accepts all *nonempty* strings with an even number of 1s. Note that an automaton accepts the empty string if and only if the start state is an accept state.

The automaton in the left of figure 2 accepts all bitstrings and the one in the right accepts no strings. The automaton in figure 3 accepts precisely the strings 01 and 0101.

Exercise 1. Design an automaton that accepts precisely the bitstrings of length at least 2 that end with 00. (See figure 4 below for a solution.)

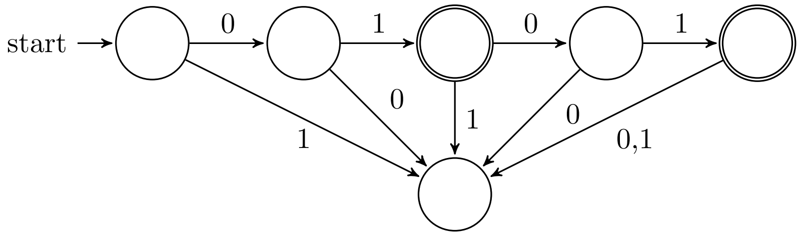


Figure 3: An automaton that accepts the strings 01 and 0101.

1.1 A mathematical definition

A *directed labeled graph* is a triple (V, Σ, E) where V and Σ are sets (called *vertices* and *labels*), and where E is a subset of $V \times V \times \Sigma$ (called *edges*). We consider graphs with self-loops, hence there are no restrictions on the set E. An element $(v, v', a) \in E$ represents an edge from $v \in V$ to $v' \in V$ with label $a \in \Sigma$. Let $w = w_1 w_2 \dots w_n \in \Sigma^*$. A *w-path* from $u \in V$ to $u' \in V$ is a sequence of edges of the form (u, v_1, w_1), (v_1, v_2, w_2), $\dots$, (v_{n-1}, u', w_n),[1] which we denote as

$$u \xrightarrow{w_1} v_1 \xrightarrow{w_2} \dots \xrightarrow{w_{n-1}} v_{n-1} \xrightarrow{w_n} u'$$

[1]In graph theory this is often called a *walk* and a *path* is defined as a walk in which all vertices are different. We use *path* in the general sense to be consistent with the term *computation path* which is commonly used in automata theory.

Definition 1. *A* deterministic finite automaton *or* DFA *is a 5-tuple (Q, Σ, E, s, F) where Q and Σ are finite sets, (called* states *and* alphabet*), $s \in Q$ (called* initial state*), $F \subseteq Q$ (called set of* accept states*), and $E \subseteq Q \times Q \times \Sigma$ such that for each $q \in Q$ and $a \in \Sigma$ there is a unique element $(q, r, a) \in E$.*[2]

The DFA accepts *$w \in \Sigma^*$ if there exists a w-path in (Q, Σ, E) from s to an element of F. The language* recognized *by the DFA, is the set of all strings that are accepted by the DFA.*

For example, the automaton at the left of figure 1 is defined by

$$(\{s, \text{Ev}, \text{Odd}\}, \{0,1\}, \left\{\begin{smallmatrix} (s, & \text{Ev}, & 0) & (s, & \text{Odd}, & 1) \\ (\text{Ev}, & \text{Ev}, & 0) & (\text{Ev}, & \text{Odd}, & 1) \\ (\text{Odd}, & \text{Odd}, & 0) & (\text{Odd}, & \text{Ev}, & 1) \end{smallmatrix}\right\}, s, \{\text{Ev}\}).$$

and recognizes the language

$$\{w : w \neq \varepsilon \text{ and } \sum_{i=1}^{|w|} w_i = 0 \bmod 2\}.$$

Definition 2. *A language is* regular *if there exists a DFA that recognizes it.*

Exercise 2.

- Show that every finite language (a language with finitely many strings) is regular.
- Show that the complement of a regular language is also regular.
- For $x = x_1 x_2 \ldots x_n \in \{0,1\}^*$ let $x_{\text{rev}} = x_n x_{n-1} \ldots x_1$. Prove that

  $$L_k = \{x : x_{\text{rev}} \text{ represents a number in binary that is a multiple of } k\}$$

  is regular for $k = 4, 3$. This exercise is harder for $n = 3$.
- Draw a schematic representation of a DFA that accepts precisely the bitstrings $x = x_1 x_2 \ldots x_n$ of length at least 2 for which $x_{n-1} = 0$. For example, the DFA should accept $101\underline{0}1$, $00\underline{0}0$, $\underline{0}1$, and reject $\varepsilon, 0, 1, \underline{1}0, 110\underline{1}1$.
- Show that if a DFA has k states, and it accepts a string of length at least k, then it accepts infinitely many strings.

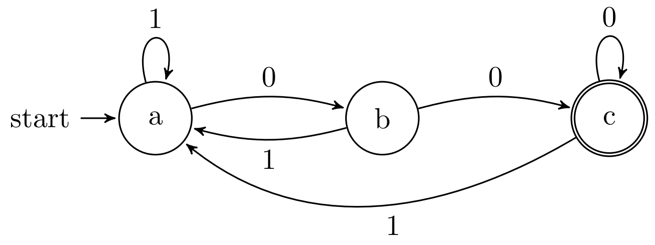


Figure 4: A solution of Exercise 1.

[2] Usually, DFAs are defined using a transition function. We use a definition that is closer to graphs, because all our proofs essentially inspect computation paths.

1.2 The pumping lemma

The pumping lemma is useful to prove that a language is not regular. *Notation.* The concatenation of two strings x and y is written as xy. For any string y, let $y^0 = \varepsilon, y^1 = y, y^2 = yy, \dots$

Lemma 1 (Pumping lemma)**.** *If L is a regular language, then there exists a number p (the pumping length) such that for all $w \in L$ of length at least p, w can be written as the concatenation xyz such that $y \neq \varepsilon$, xy has length at most p, and $xy^iz \in L$ for all $i = 0, 1, 2, \dots$.*

Note that we cannot omit the condition $y \neq \varepsilon$, otherwise the theorem is trivially true.

Proof. If an automaton with p states has processed p letters of its input, there have been $p+1$ placements of the finger. Hence, there is a state in which the finger has been placed twice or more. Let y be the substring of the input that was red between two such placements. Note that y is not empty. It is easy to observe that removing y from w also yields a strings that is accepted. On the other hand, repeating y any number of times also yields a string that is accepted.

To get familiar with the notation of Definition 1, we reformulate the argument in a more mathematical language. Assume an automaton (Q, Σ, E, s, F) with p states accepts a string $w = w_1 \dots w_n$ with $n \geq p$. Then there exists a w-path

$$q_0 \xrightarrow{w_1} q_1 \xrightarrow{w_2} \dots \xrightarrow{w_n} q_n$$

with $q_0 = s$ and $q_n \in F$. Among $q_0, q_1, q_2, \dots, q_p$ there is a state that appears at least twice. Assume $q_i = q_j$ for $i < j \leq p$. Then the path

$$q_0 \xrightarrow{w_1} q_1 \xrightarrow{w_2} \dots \xrightarrow{w_i} q_i \xrightarrow{w_{j+1}} q_{j+1} \xrightarrow{w_{j+2}} \dots \xrightarrow{w_n} f$$

shows that the automaton accepts $w_1 \dots w_i w_{j+1} \dots w_n$. Let $x = w_1 \dots w_i$, $y = w_{i+1} \dots w_j$ and $z = w_{j+1} \dots w_n$. We have shown that the automaton accepts xz. In a similar way it follows that the automaton accepts xy^kz for all $k = 0, 1, 2 \dots$ $\square$

Lemma 2. *The language $\{0^n1^n | n \geq 0\}$ is not regular.*

Proof. We prove this by contradiction. Suppose that the language is regular. Let p be its pumping length. We apply the pumping lemma to the string 0^p1^p, and obtain x, y and z with

$$xyz = 0^p1^p, \qquad \text{and} \qquad xy \text{ has length at most } p,$$

This implies that x and y only contain zeros. For all i the strings xy^iz also belong to the language. These strings are of the form $0^n0^{\ell(i-1)}1^n$, where ℓ is the length of y. Note that $\ell \geq 1$ because of the condition $y \neq \varepsilon$ in the pumping lemma. Thus for $i \neq 1$, the string does not belong to the language. A contradiction. Our initial assumption must be wrong: the language is not regular. $\square$

Exercise 3. Show that the following languages over $\{0, 1\}$ are not regular:

- $\{w | w$ has an equal number of 0's and 1's $\}$
- $\{1^{n^2} | n \geq 0\}$

- $\{0^i 1^j | i > j\}$ (hint: use "pumping down").

Exercise 4. Give examples of languages A and B such that:

1. $A \subseteq B$, A is regular and B is not regular.
2. $A \subseteq B$, A is not regular and B is regular.

1.3 The class of regular languages is closed under unions and intersections

Exercise 5. Make a schematic representation of a DFA that recognizes the set of strings in $\{a, b\}^*$ that have even length and for which the number of symbols a is a multiple of 3.

Theorem 3. *If A and B are regular languages, then $A \cap B$ is also a regular language.*

Note that this implies that $A \setminus B$ are also regular, because regular languages are closed under complements, as we know from exercise 2.

Proof. We construct a new machine that simulates the DFAs in parallel in a similar way as in exercise 5. Let (Q, Σ, E, s, F) and (Q', Σ, E', s', F') be DFAs recognizing A and B. The new DFA is $(\tilde{Q}, \Sigma, \tilde{E}, \tilde{s}, \tilde{F})$ where

$$\begin{aligned}\tilde{Q} &= Q \times Q' \\ \tilde{s} &= (s, s') \\ \tilde{F} &= F \cap F',\end{aligned}$$

and

$$\tilde{E} = \big\{\big((q, q'), (r, r'), a\big) : (q, r, a) \in E \text{ and } (q', r', a) \in E'\big\}$$

We show that these automata accept the same strings: there exists a w-path from (s, s') to $(f, f') \in \tilde{F}$ if and only if there exist w-paths from s to $f \in F$ and s' to $f' \in F'$ in the automata for A and B; because, if there exists a w-path

$$(s, s') \xrightarrow{w_1} (q_1, q_1') \xrightarrow{w_2} \dots \xrightarrow{w_{n-1}} (q_{n-1}, \tilde{q}_{n-1}) \xrightarrow{w_n} (f_n, f_n')$$

then there exist w-paths

$$\begin{aligned} s &\xrightarrow{w_1} q_1 \xrightarrow{w_2} \dots \xrightarrow{w_{n-1}} q_{n-1} \xrightarrow{w_n} f_n \\ s' &\xrightarrow{w_1} q_1' \xrightarrow{w_2} \dots \xrightarrow{w_{n-1}} q_{n-1}' \xrightarrow{w_n} f_n', \end{aligned}$$

and vice versa. Hence, if w is accepted by the new machine, w will be accepted by the machines for A and B and vice versa. $\square$

Exercise 6. With a small modification of the proof, it can be shown that if A and B are regular, then $A \cup B$ is also regular. What is this modification?

Exercise 7. By the previous exercise, we conclude that the regular languages are closed under finite unions. This is not true for an infinite sequence of unions. Give an infinite sequence of regular languages $L_1, L_2, \dots$ such that $L_1 \cup L_2 \cup \dots$ is not regular.

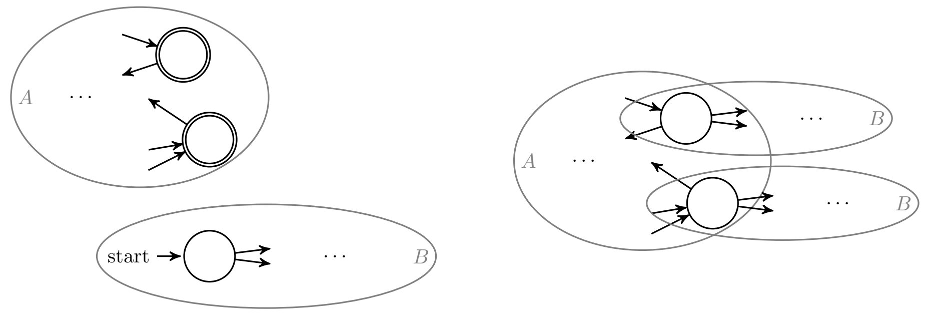


Figure 5: "Concatenating" automata does not work to concatenate regular languages.

Definition 3. *For $x, y \in \Sigma^*$ let $xy \in \Sigma^*$ be the concatenation of x and y. For $A, B \subseteq \Sigma^*$ let:*
- $AB = \{xy : x \in A \text{ and } y \in B\}$,
- $A^* = \{\varepsilon\} \cup A \cup AA \cup \ldots$

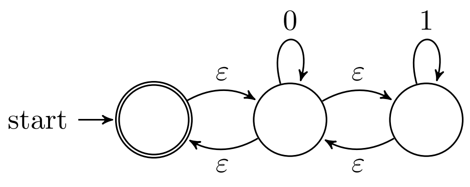
In the next section we prove that the concatenation AB of two regular languages A and B is also regular. However, this is not obvious. We explain the problem. Let $w \in AB$ and let $w = uv$ such that $u \in A$ and $v \in B$. Given automata for A and B we might try to design an automaton that follows the A-automaton while scanning u and then "transfers" to the B-automaton to scan v, as illustrated in figure 5: we make a copy of the B-automaton for each accept state of the A-automaton and "glue" it to the start state of the B-automaton. But this does not work: for example, the glued state has now 2 outgoing edges for each symbol.

In the next section we study automata for which there is no condition on the number of outgoing edges.

2 Nondeterministic finite automata

In computational complexity, nondeterministic computation plays an important role. Now we define a nondeterministic variant of finite automata and show that these automata recognize the same class of languages.

A nondeterministic finite automata consists of the same components as a deterministic one. There are two changes: firstly, the restriction on the number of outgoing edges from a state with a fixed label is dropped: there might be zero, one or more such outgoing edges. Secondly, some arrows are labelled by the symbol $\varepsilon \notin \Sigma$. A nondeterministic automata *accepts* a string $w = w_1 \ldots w_n$ if it is possible to place a finger in the start state and then move it to an accept state as follows: first it moves through any number of ε-arrows, then through a w_1-arrow, then through any number of ε-arrows, then through a w_2-arrow, ..., then through a w_n arrow, and finally through any number of ε-arrows to reach the accept state. For example, the following (rather strange) automaton accepts all bitstrings.



Another example is the automaton from figure 8 which accepts all strings of length at least two for which the one but last symbol equals 0, it is, the strings $w = w_1 \dots w_n$ for which $w_{n-1} = 0$. (For example, it accepts $\underline{0}0$, $\underline{0}1$, and $111\underline{0}1$; and rejects ε, 1, $\underline{1}0$, $0\underline{1}0$ and $001\underline{1}1$.)

The languages accepted by non-deterministic automata are closed under the union operation (if A and B are recognized by nondeterministic automata, then also $A \cup B$ is recognized by such an automaton). This is illustrated in figure 2 for A and B in $\{a, b\}^*$, being the sets of strings with an even number of a's and b's.

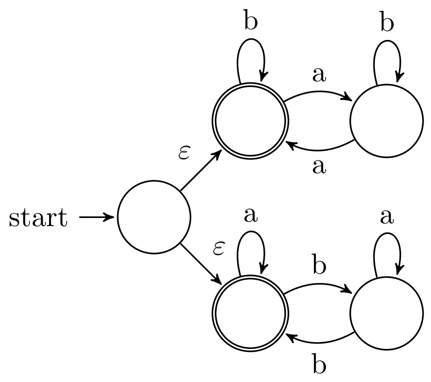


Figure 6: An NFA that recognizes the strings with an even number of a's or an even number of b's.

2.1 Definition

Definition 4. *A* nondeterministic finite automata *or* NFA *is a 5-tuple* (Q, Σ, E, s, F) *where* Q *and* Σ *are finite sets,* $\varepsilon \notin \Sigma$, $E \subseteq Q \times Q \times (\Sigma \cup \{\varepsilon\})$, $s \in Q$ *and* $F \subseteq Q$.

An NFA accepts $w \in \Sigma^*$ *if there exists a path from* s *to an element of* F *in the graph* $(Q, \Sigma \cup \{\varepsilon\}, E)$ *for which the concatenation of the edges' labels equals* w*; in this concatenation symbols* ε *represent empty strings (such paths are also called* w-paths*).*

The pumping lemma also holds for non-deterministic automata. If 2 languages A and B are recognized by NFAs, then AB is also recognized by some NFA: we simply connect the accept states of the A-automaton to the start state of the B automatone using ε-edges (these are edges labelled with ε), see figure 7.

Exercise 8. Suppose that in an NFA no ε-edges leave from the start state. Show that there exists an NFA without ε-edges that recognizes the same language.

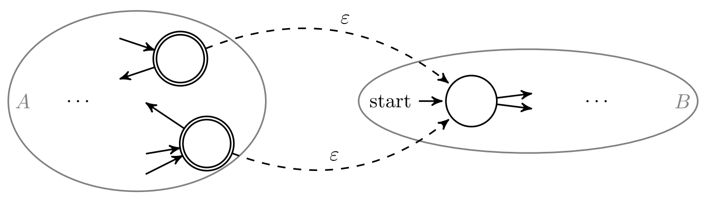


Figure 7: "Concatenating" NFAs.

Exercise 9. Suppose a DFA recognizes a language L. If all accepting states are changed into rejecting states, and all rejecting states are changed into accepting states, then the new DFA recognizes the complement $A^* \setminus L$ of the language L. Construct an NFA for which this is not true.

In the next section we use the following exercise.

Exercise 10. Show that for each NFA there exists an NFA that

- recognizes the same language,
- the start state has no incoming edges,
- there is exactly one accept state,
- the accept state has no outgoing edges.

2.2 Checking whether an NFA accepts a string

Let $|w|$ be the length of w and $|Q|$ be the number of elements in the set Q. To decide whether an NFA accepts w, one needs to search through all paths in the graph. Although one only needs to search for paths of length at most $|Q||w|$, the search tree can have exponential size. We present a more efficient algorithm that runs in time $O(|E||w|)$ (in the RAM-model). This algorithm will be used to prove that NFAs and DFAs recognize the same languages.

Proposition 4. *There exists an algorithm that on input a string w and a description of an NFA with m states, determines whether the NFA accepts w in time $O(|w|m^2)$.*

For some NFA (Q, Σ, E, s, F), let δ be the function that maps a string $w \in \Sigma^*$ to the set of all endpoints of w-paths:

$$\delta(w) = \{q \in Q : \text{ there exists a } w\text{-path from } s \text{ to } q\}.$$

(In Sipser's book the notation δ^* is used.) The following inductive definition of δ is equivalent: let $a \in \Sigma$ and $w \in \Sigma^*$

$$\begin{aligned} \delta(\varepsilon) &= \{q : \text{ there exists an } \varepsilon\text{-path from } s \text{ to } q\} \\ \delta(wa) &= \left\{q' : \text{ there exists } q \in \delta(w) \text{ and } a\text{-path from } q \text{ to } q' \right\}. \end{aligned}$$

Equivalently, we could replace the second condition by the following detailed condition.

$$\delta(wa) = \left\{q'' : \text{there exists } q \in \delta(w),\, q' \text{ such that } (q, q', a) \in E, \text{ and an } \varepsilon\text{-path from } q' \text{ to } q'' \right\}.$$

Given a set of vertices in a graph, we can compute the set of all endpoints of ε-edges in time $O(|E|)$. For this we can use the methods called dept-first search or breath-first search. We explain the latter: let $S = \{s\}$, replace S by $S \cup \bigcup_{q \in S} \{r : (q, r, \varepsilon) \in E\}$, and repeat this replacement until S does no longer change. We can do this in a way that every edge is visited at most twice: every vertex in S needs to be inspected at most once and every edge is incident on precisely 2 vertices. Hence, the algorithm runs in time $O(|E|) \leq O(|Q|^2)$.

```
**Data**: $w \in \Sigma^*$
**Result**: $\delta(w) \subseteq Q$
if $w = \varepsilon$ then
    Output $\delta(\varepsilon)$.
else
    Let $w = va$ with $v \in \Sigma^*$ and $a \in \Sigma$.
        • Recursively compute $\delta(v)$.
        • Compute the set $S$ of all endpoints of $a$-labelled edges leaving from $\delta(v)$.
        • Output the set of all endpoints of $\varepsilon$-paths starting in $S$.
```

The two last lines can be evaluated in time $O(E)$ and the number of recursive calls equals $|w| + 1$. Hence, the algorithm runs in time $O(|w||E|)$.

In figure 9 the evaluation of the algorithm above is illustrated for the automaton of figure 8 and input 0100. Note that for a fixed automaton, the algorithm uses constant space, (for example, the set S can be stored in a binary vector of size $|Q|$), and hence can be simulated by a DFA. This will be the main idea in the next subsection.

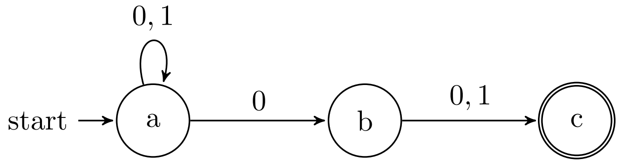


Figure 8: A nondeterministic automaton that recognizes strings with a 0 in the one but last position.

$$\begin{tikzcd}
\delta(\varepsilon) & & & & & a \arrow[dll] \arrow[dr] & & \{a\} \arrow[d, "0"] \\
\delta(0) & & & a \arrow[d] & & & b \arrow[d] & \{a,b\} \arrow[d, "1"] \\
\delta(01) & & & a \arrow[dl] \arrow[dr] & & & c & \{a,c\} \arrow[d, "0"] \\
\delta(010) & & a \arrow[dl] \arrow[dr] & & b \arrow[d] & & & \{a,b\} \arrow[d, "0"] \\
\delta(0100) & a & & b & c & & & \{a,b,c\}
\end{tikzcd}$$

Figure 9: The computation tree for the automaton from figure 8 on input 0100.

2.3 NFAs and DFAs recognize the same languages

Theorem 5. *A language is regular if and only if some NFA recognizes it.*

Proof. Every DFA is an NFA, and the $\Rightarrow$ direction of the theorem is trivial. The $\Leftarrow$ direction follows by noticing that we can "evaluate" the function δ with a DFA. Below the proof an example is given.

We need to show that for every NFA (Q, Σ, E, s, F) there exists a DFA (Q', Σ, E', s', F') that recognizes the same language. Each state in Q' is a subset of Q. The machine is given by:

$$\begin{aligned} Q' &= \{\delta(w) : w \in \Sigma^*\} \\ E' &= \{(\delta(w), \delta(wa), a) : a \in \Sigma \text{ and } w \in \Sigma^*\} \\ s' &= \delta(\varepsilon) \\ F' &= \{q' \in Q' : \ q' \text{ contains an element from } F\}. \end{aligned}$$

We verify that the condition on the outgoing edges for a DFA is satisfied. Let $(q', a) \in Q' \times \Sigma$. By definition of Q' there is a string w such that $\delta(w) = q'$. Hence, there is at least one edge $(q', \delta(wa), a)$. By the recursive definition of δ, the value of $\delta(va)$ is fixed for all v with the same value $\delta(v)$. By choice of E', all outgoing edges from q' are of the form $(\delta(v), \delta(va), a)$ with $\delta(v) = q'$. Thus, the value $\delta(va)$ is also fixed, and the condition on the outgoing edges is satisfied.

For each $w = w_1 \ldots w_n$, there is precisely one w-path in the DFA leaving from s:

$$\delta(\varepsilon) \xrightarrow{w_1} \delta(w_1) \xrightarrow{w_2} \delta(w_1 w_2) \xrightarrow{w_3} \ldots \xrightarrow{w_n} \delta(w_1 \ldots w_n).$$

Hence, w is accepted by the DFA if and only if $\delta(w_1 \ldots w_n) \in F'$, if and only if there is a w-path from s to an element in F, if and only it is accepted by the NFA. □

The proof gives us an effective way to transform an NFA to a DFA. We illustrate this with the automaton of figure 8 (which recognizes strings with a 0 in the one but last position). This NFA has states a, b, c, the states of the new machine corresponds to subsets of this set. The start state is $\{a\}$ and $\delta(\varepsilon) = \{a\}$. Notice that $\delta(0) = \{a, b\}$. More general, if $\delta(x) = \{a\}$, then $\delta(x0) = \{a, b\}$. We place a 0-arrow from the state $\{a\}$ to the state $\{a, b\}$, see figure 2.3. If $\delta(x) = \{a\}$, then $\delta(x1) = \{a\}$, and we place a 1-arrow from $\{a\}$ to itself. If $\delta(x) = \{a, b\}$, then $\delta(x0) = \{a, b, c\}$; we place a 0-arrow from $\{a, b\}$ to $\{a, b, c\}$, etc. We repeat this until all states have leaving 0 and 1 arrows. Finally, the accept states of our automaton are the states that contain an element c.

Corollary 6. *If A and B are regular, then AB and A^* are regular.*

Exercise 11. Prove Corollary 6.

Exercise 12. For a language L, let $L_{\text{rev}} = \{w : w_{\text{rev}} \in L\}$. Show that if L is regular, then also L_{rev} is regular.

A unary language is a language over an alphabet that contains only one symbol. For example, $\{0^{2n} : n \in \mathbb{Z}_{\geq 0}\}$ is a unary language over $\{0\}$.

Exercise 13. Show that for large n there exists a unary language that is recognized by an NFA of size at most n^3 but is not recognized by a DFA of size less than 2^n.
You can use that the mth prime is less than $m(\ln m + \ln m \ln m)$ for all $m \geq 6$.[3]

[3]The result is inequality 3.13 in the corollary on page 69 in: Rosser, J.B., and Schoenfeld, L. Approximate formulas for some functions of prime numbers. *Illinois Journal of Mathematics* 6.1 (1962): 64-94. A related result is the prime number theorem: *the number $\pi(n)$ of primes less or equal to n satisfies:* $\lim_{n\to\infty} \frac{\pi(x)}{n/\ln n} = 1$.

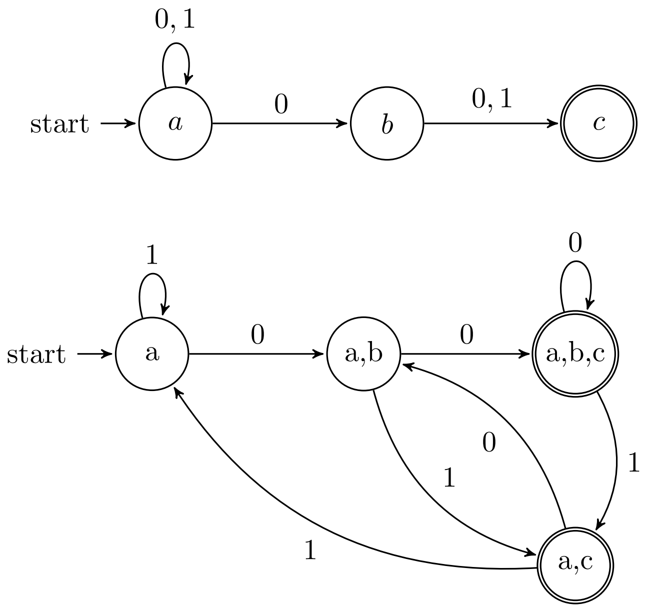


Figure 10: Transformation of an NFA to a DFA.

3 Regular expressions

3.1 Definition

Regular expressions are often used in the design of compilers. They are defined by induction.

Definition 5. *R is a regular expression over an alphabet Σ if and only if*

- *$R = a$ for some $a \in \Sigma$,*
- *$R = \varepsilon$,*
- *$R = \emptyset$,*
- *$R = \Sigma$,*
- *$R_1 \cup R_2$ where R_1 and R_2 are regular expressions,*
- *$R_1 R_2$ where R_1 and R_2 are regular expressions,*
- *R^* where R is a regular expression.*

The language 'R' modelled *by R is also defined by induction: 'a' $= \{a\}$, 'ε' $= \{\varepsilon\}$, '$\emptyset$' $= \emptyset$, 'Σ' $= \Sigma$, '$R_1 \cup R_2$' $=$ 'R_1' $\cup$ 'R_2', '$R_1 R_2$' $=$ 'R_1''R_2' and 'R^*' $=$ 'R'*. R* models *w if $w \in$ 'R'.*

Often, regular expression are associated to the language they model, and one writes R in stead of 'R'. One should be careful, writing a statement $0 \in$ '0' as $0 \in 0$ might lead to logical problems.

Note that '$R \cup \emptyset$' $=$ 'R', '$R\varepsilon$' $=$ 'R', and '$R\emptyset$' $=$ '$\emptyset$'. The $\cup$ operator has lowest priority, and the star operator has priority over concatenation. For example: '$a \cup bc^*$' $= \{a, b, bc, bcc, bccc, \dots\}$ and '$a \cup (bc)^*$' $= \{a, \varepsilon, bc, bcbc, \dots\}$. Some examples for $\Sigma = \{0, 1\}$:

- $0^*10^* \leftrightarrow$ strings with exactly one 1,
- $\Sigma^*1\Sigma^* \leftrightarrow$ strings with at least one 1,
- $(\Sigma\Sigma)^* \leftrightarrow$ strings of even length,
- $01 \cup 10 \leftrightarrow \{01, 10\}$,
- $(0 \cup \varepsilon)(1 \cup \varepsilon) \leftrightarrow \{\varepsilon, 0, 1, 01\}$,
- $1^*\emptyset \leftrightarrow \emptyset$,
- $\emptyset^* \leftrightarrow \{\varepsilon\}$.

Exercise 14. Let $\Sigma = \{0, 1\}$. Which of the following languages are regular?

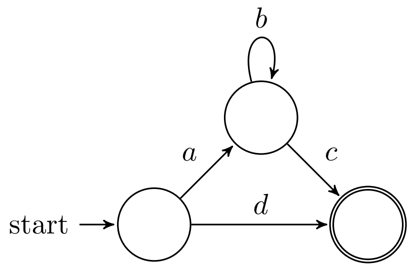
$$\bigcup_{n\geq 1} \text{`}\Sigma^*0^n1^n\Sigma^*\text{'} \qquad \text{and} \qquad \bigcup_{n\geq 1} \text{`}\Sigma^*0^n1^n\text{'}.$$

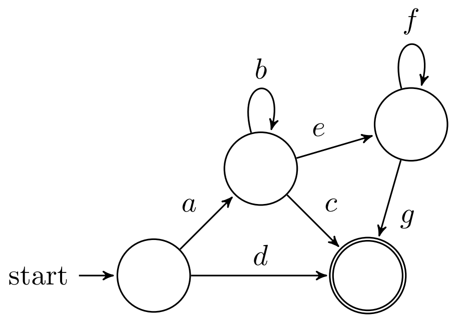
3.2 Equivalence with regular languages

Theorem 7. *A language is regular if and only if it is modeled by a regular expression.*

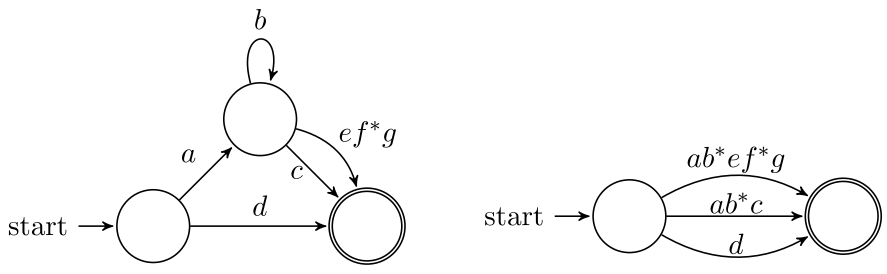
Proof of the $\Leftarrow$ direction. The automata for $R = \varepsilon$, $R = \emptyset$, $R = \Sigma$ and $R = \{a\}$ for all $a \in A$ are easy to define. For the other expression, this follows by induction from the results above, because regular languages are closed under union, concatenation and the star operation. □

Exercise 15. What are the regular expressions for the languages over $\{a, b, c, d, e, f, g\}$ recognized by the automata





To show that there exists a regular expression for the language recognized by an NFA, we transform an automaton in small steps to a regular expression. For this we need a hybrid form of regular expressions and automata. A *generalized* nondeterministic finite automaton or *GNFA* is schematically represented in the same way as an NFA, but now each arrow is labeled by a regular expression. As usual, the input is scanned from left to right, and while following an arrow, we select consecutive substrings from the input string that are modelled by the arc's regular expression. For example, the automaton at the right in exercise 15 recognizes the same language as



Automata equivalent to the automaton at the right of exercise 15.

Note that any NFA can be directly interpreted as a GNFA because an arrow labelled by an element from $\Sigma \cup \{\varepsilon\}$ has an obvious interpretation as a regular expression.

Let $\mathcal{R}_\Sigma$ denote the set of all regular expressions over Σ.

Definition. *A* GNFA *is a* 5*-tuple* $(Q, \mathcal{R}_\Sigma, E, s, F)$ *with* $s \in Q$*,* $F \subseteq Q$*, and* E *is a finite subset of* $Q \times Q \times \mathcal{R}_\Sigma$*. An* R-path *in the graph* $(Q, \mathcal{R}_\Sigma, E)$ *is a path for which the concatenation of the edge's regular expressions equals* R*. The GNFA* accepts $w \in \Sigma^*$ *if there exist an* $R \in \mathcal{R}_\Sigma$ *and an* R*-path from* s *to an element of* F *in* $(Q, \mathcal{R}_\Sigma, E)$ *such that* R *models* w*.*

Exercise 16. Show that a language is regular if and only if it is recognized by a GNFA.

At the end of the proof we need an additional property.

Definition. *A GNFA is* good *if it has (1) a unique accept state, (2) this state differs from the start state and has no outgoing edges, and (3) the start state has no incomming edges.*

Note that every GNFA is equivalent to a good one: change all accept states into normal states, add a new accept state, and connect the old accept states using ε-edges. Similar for the start state. See also exercise 10.

Lemma 8. *Every good GNFA with* $m \geq 3$ *states is equivalent to some good GNFA with* $m-1$ *states.*

We first explain why the lemma implies the theorem.

Proof of the $\Rightarrow$ *direction of Theorem 7.* We transform an NFA to an equivalent good GNFA as explained above. By iterative application of Lemma 8, we obtain an equivalent good GNFA with 2 states. By definition of good GNFAs, these states are the start and accept states, and all edges go from the start to the accept state, (the same is true in the right of figure 11). This GNFA is equivalent to the union of the regular expressions of all edges. $\square$

Proof of Lemma 8. Consider a good GNFA $(Q, \mathcal{R}_\Sigma, E, s, \{f\})$ and an $r \in Q \setminus \{f, s\}$. Because $|Q| \geq 3$, such r exists. Let $G = \varepsilon$ if r has no loops, and otherwise, let G be the union of the expressions of all loops on r. We show that the GNFA

$$(Q_r, \mathcal{R}_\Sigma, E_1 \cup E_2, s, \{f\})$$

with $Q_r = Q \setminus \{r\}$ and

$$E_1 = E \cap (Q_r \times Q_r \times \mathcal{R}_\Sigma)$$
$$E_2 = \big\{(p, q, FG^*H) : p, q \in Q_r \text{ and } (p, r, F), (r, q, H) \in E\big\}.$$

is an equivalent good GNFA. This implies the lemma.

The GNFA is good: the start and accept states are s and f, and they are different because the original automaton is good. Moreover, E_2 contains only outgoing edges in states that have outgoing edges in E, hence f has no outgoing edges. For similar reasons, s has no incomming edges. It remains to prove that the GNFAs are equivalent:

$$\exists R \in \mathcal{R}_\Sigma \text{ that models } w\text{, and an } R\text{-path from } s \text{ to } f \text{ in } (Q, \mathcal{R}_\Sigma, E)$$
$$\Updownarrow$$
$$\exists \tilde{R} \in \mathcal{R}_\Sigma \text{ that models } w \text{ and a } \tilde{R}\text{-path from } s \text{ to } f \text{ in } (Q_r, \mathcal{R}_\Sigma, E_1 \cup E_2)\,.$$

Proof of $\Downarrow$-direction. Let the R-path be

$$q_0 \xrightarrow{R_1} q_1 \xrightarrow{R_2} \dots \xrightarrow{R_{n-1}} q_{n-1} \xrightarrow{R_n} q_n.$$

If $q_i \neq r$ for all $i \leq n$, then all the edges of the path are in E_1, and the conditions are satisfied for the same path and $\tilde{R} = R$. Otherwise, let i and $k \geq 1$ be such that the path contains a part

$$q_{i-1} \xrightarrow{R_i} r \xrightarrow{R_{i+1}} \dots \xrightarrow{R_{i+k-1}} r \xrightarrow{R_{i+k}} q_{i+k},$$

with $q_{i-1} \neq r$ and $q_{i+k} \neq r$. We replace this part by

$$q_{i-1} \xrightarrow{\quad R_i G^* R_{i+k} \quad} q_{i+k+1},$$

which is an edge in $E_1 \cup E_2$ by construction of E_2. Every string modeled by $R_{i+1}R_{i+2}\dots R_{i+k-1}$ is also modelled by G^* by definition of G. Because w was modelled by the concatenated labels of the oringal path, this is also true after the modification.

We repeat this until all states r are removed and obtain a path with edges in $E_1 \cup E_2$. Moreover, the concatenated labels $\tilde{R}$ model w and the endpoints are the same.

Proof of the $\Uparrow$-direction. Let the $\tilde{R}$-path with edges in $E_1 \cup E_2$ be

$$q_0 \xrightarrow{\tilde{R}_1} q_1 \xrightarrow{\tilde{R}_2} \dots \xrightarrow{\tilde{R}_{n-1}} q_{n-1} \xrightarrow{\tilde{R}_n} q_n.$$

If all $\tilde{R}_i \in E_1$, then this is also a path in $(Q, \mathcal{R}_\Sigma, E)$. Otherwise, assume that $\tilde{R}_i \in E_2$. Let u be the substring of w that is modeled by $\tilde{R}_i$ and let $p \xrightarrow{\tilde{R}_i} q$ be the corresponding edge. Thus, u is modelled by $FG_1G_2\dots G_kH$ for some expressions that satisfy

$$(p, r, F), (r, r, G_1), \dots, (r, r, G_k), (r, q, H) \in E.$$

In the path, we replace the edge $p \xrightarrow{R_i} q$ by the edges

$$p \xrightarrow{F} r \xrightarrow{G_1} \dots \xrightarrow{G_k} r \xrightarrow{H} q,$$

which all belong to E. We can do this for all $\tilde{R}_i$ in E_2, and obtain a path with edges in E. Moreover, the concatenation R of the labels models w and the endpoints are the same. $\square$

Exercise 17. Apply the transformation in the proof to the automata in figure 12 to obtain an equivalent regular expression.

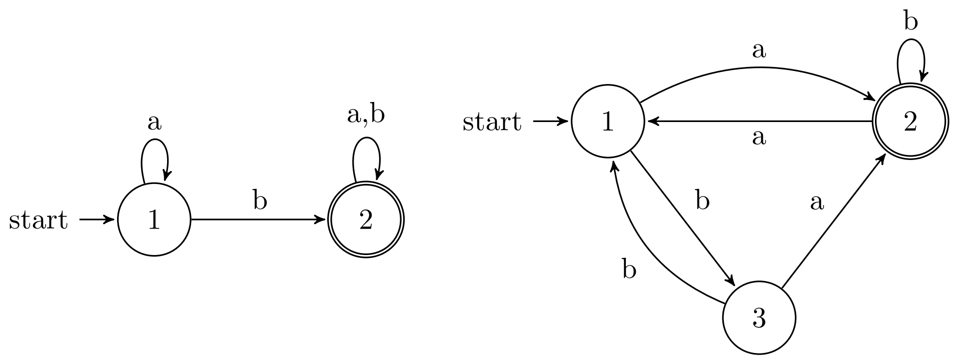


Figure 12: Automata for exercise 17.