

Automata, Computability & Complexity

Author: Daria Shutina

Automata, Computability & Complexity

23-02-06

Org stuff

von Neumann architecture

Finite automata

Deterministic finite automata

Definitions

Example 1: a STD

Example 2: finding FA

Example 3: finding a FA for a union of RLs

Regular language

Regular operations

Th 1.1 (union operation of RLs)

Th. 1.2 (concatenation operation of RLs)

23-02-13

Nondeterministic finite automata

Definitions

Example of NFA

Creating a tree for an input

Th 2.1 (equivalence between NFA and FA)

Corollary 2.1 (relation between RL and NFA)

Th 2.2 (star operation of RLs)

23-02-20

Regular expressions

Lemma 3.1

Example: building an NFA

GNFA: generalized nondeterministic finite automaton

Definition

Example

Th. 3.1 (equivalence between FA and GNFA)

Example

Lemma 3.3 \todo

Th.

23-02-27

Nonregular Languages

the Pumping Lemma

Example 1

Example 2 \TODO

Context-Free Grammars

23-02-06

Org stuff

For schemes: https://www.cs.unc.edu/~otternes/comp455/fsm_designer/

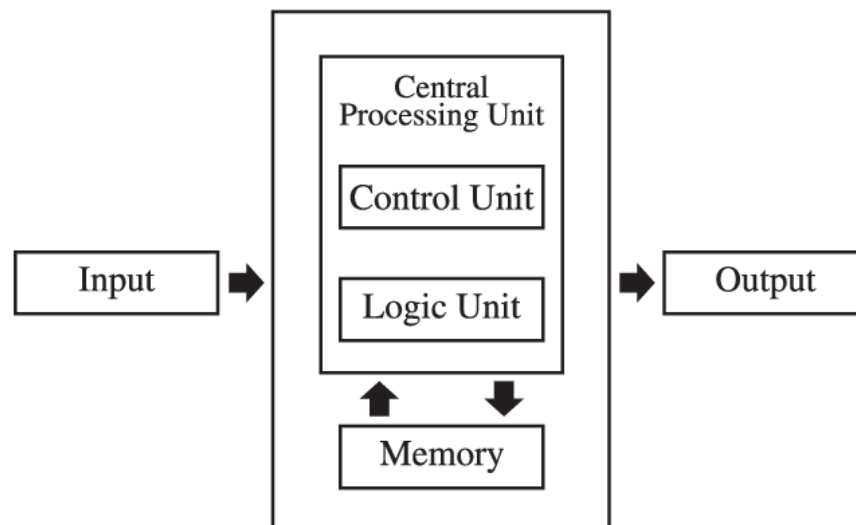
Timetable:

- Mondays 14:15 lectures offline. Moodle quiz in the beginning of each lecture
- Wednesdays 14:15 lectures offline
- Thursdays 17:15 consultations online. Need to book

Grade:

- 100% exam
- bonus 0.33 (5%) if 50% of quizzes and 50% of homeworks are done

von Neumann architecture



Our nowadays computers are based on the von Neumann computer architecture.

Finite automata

For theory achievements, we will use a simpler compute model which gets the input as a string of symbols, has no memory, and generates an output that is either `accept` or `reject`. This machine is called *finite automaton*.

Deterministic finite automata

Deterministic means that an input symbol a leads from state q to exactly one possible state q' .

The visualization of the input computation is a chain.

Definitions

Definition 1.1 (Finite automaton) A **finite automaton** (FA) M is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$ where

1. Q is a finite set called the **states**,
2. Σ is a finite set called the **alphabet**,
3. $\delta : Q \times \Sigma \rightarrow Q$ is the **transition function**,
4. $q_0 \in Q$ is the **start state**, and
5. $F \subseteq Q$ is the **set of accept states / final states**.

A transition function can be described via *state transition diagram* (STD) or *state transition table*.

If a state is a *start state*, it has an arrow pointing from nowhere.

What is an *accept state*? Imagine we have an input. We go from one state to another, as the input says. Then we finish in a definite state. This state can be a final/accept state. In other words, it is where an input ends. Final states are shown as double-circled states in STD.

FA accepts a string if it starts in a start state, uses only transitions of δ and finishes in one of final states.

Definition 1.2 (Strings accepted by M) Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and $w = w_1w_2 \cdots w_n$ be a string over alphabet Σ .

M **accepts** w if there exists a sequence of states r_0, r_1, \dots, r_n , such that all following three conditions hold:

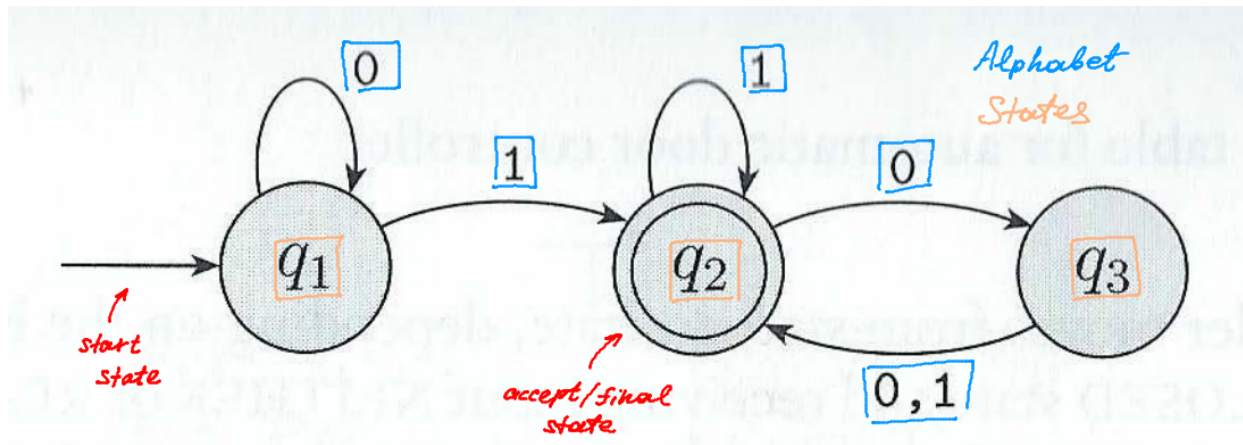
1. $r_0 = q_0$ (M starts in start state.)
2. $\delta(r_i, w_{i+1}) = r_{i+1}$, for $i = 0, \dots, n-1$
(State change follows transition function.)
3. $r_n \in F$ (M ends up in accept state)

If M does not accept w , it **rejects** it.

A *computation of FA on a string* is a sequence of states such that it starts in a start state and uses only transitions of δ .

$L(M)$ -- the language of machine M -- is the set of all strings that are accepted by M . Every FA still recognizes an empty language \emptyset .

Example 1: a STD



$$Q = \{q_1, q_2, q_3\}; \Sigma = \{0, 1\}; F = \{q_2\}.$$

q_1 is a start state.

δ can be described with a table:

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

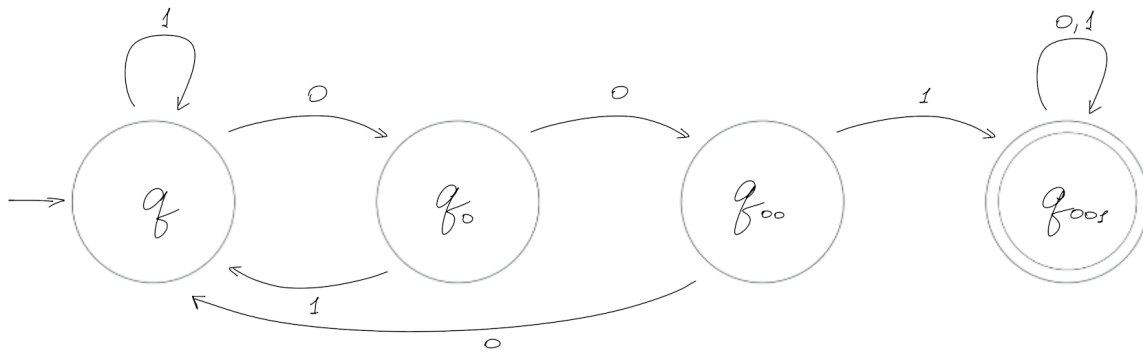
FA accepts a string 1101, for example, since it starts in q_1 and finishes in q_2 .

Example 2: finding FA

We consider a language $L = \{w \mid w \text{ contains } 001 \text{ as substring}\}$. The alphabet is $\{0, 1\}$.

The idea is that we will have 4 states:

- q -- no subsequence
- q_0 -- we have 0
- q_{00} -- we have 00
- q_{001} -- we have 001 and we need to stop.



Example 3: finding a FA for a union of RLs

$$M_1 = (S, \Sigma_1, \delta_1, q_1^0, F_1)$$

$$M_2 = (T, \Sigma_2, \delta_2, q_2^0, F_2)$$

We can find a new FA for $M = (Q, \Sigma, \delta, q_0, F)$ where

$$Q = S \times T$$

$$\Sigma = \Sigma_1 \cup \Sigma_2$$

$$\delta((s, t), a) = (\delta_s(s, a), \delta_t(t, a))$$

$$q_0 = (s_0, t_0)$$

$$F = F_1 \cup F_2$$

Regular language

A language is called a *regular language* if \exists FA that recognizes it.

To prove that a language is regular, we need to build a FA that will recognize it. If we are able to build a STD, then the language is regular.

Example of a non-regular language:

$$L = \{0^n 1^n \mid n \geq 1\}.$$

n is not fixed, so we have a problem with choosing a transition function.

Regular operations

Definition 1.6 (Regular operations) Let A and B be languages. We define the regular operations **union**, **concatenation** and **star** as follows:

- **Union:** $A \cup B := \{x \mid x \in A \text{ or } x \in B\}$
- **Concatenation:** $A \circ B := \{xy \mid x \in A \text{ and } y \in B\}$.
- **Star:** $A^* := \{x_1x_2 \dots x_k \mid k \geq 0 \text{ and each } x_i \in A\}$.

Th 1.1 (union operation of RLs)

The class of regular languages is closed under the union operation:

A_1, A_2 are RLs $\Rightarrow A_1 \cup A_2$ is a RL.

Proof:

There are regular languages A_1 and A_2 . W

There exists FAs M_1 and M_2 such that

$$\begin{cases} L(M_1) = A_1, & M_1 = (S, \Sigma, \delta_1, s_0, F_1) \\ L(M_2) = A_2, & M_2 = (T, \Sigma, \delta_2, t_0, F_2) \end{cases}$$

If M_1 and M_2 have different alphabets, then Σ will be the union of their alphabets.

To show that $A_1 \cup A_2$ is a RL, we construct a FA M such that $M = (Q, \Sigma, \delta, q_0, F)$

$$Q = S \times T$$

Σ is the same

$$\delta((s, t), a) = (\delta_s(s, a), \delta_t(t, a))$$

$$q_0 = (s_0, t_0)$$

$$F = F_1 \times F_2$$

Th. 1.2 (concatenation operation of RLs)

The class of regular languages is closed under the concatenation operation.

A_1, A_2 are RLs $\Rightarrow A_1 \circ A_2$ is a RL.

Proof: его нет

23-02-13

Nondeterministic finite automata

In contrast to deterministic FAs, nondeterministic FAs allow several successor states (or even none) for a given fixed input. An NFA adds more flexibility to the computation.

- There can be several transitions with the same symbol
- There can also be no transition for some symbol
- There can be additional label ϵ . It is like a special symbol (i.e., an empty string). Every alphabet has its own ϵ (provided that it has special symbols),

The visualization of the computation is a tree.

Definitions

Definition 1.7 (Nondeterministic finite automaton) A **nondeterministic finite automaton** (NFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set of states,
2. Σ is a finite alphabet,
3. $\delta : Q \times \Sigma_\epsilon \rightarrow \mathcal{P}(Q)$ is the transition function, with $\Sigma_\epsilon := \Sigma \cup \{\epsilon\}$,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

In the transition function definition, $\mathcal{P}(Q)$ means a set of states, since in NFA we can get from one state to several (or zero) states.

NFA *accepts* a given input string, if there exists a computation branch in a tree that ends in an accept state; otherwise it *rejects* it.

Definition 1.8 (Strings accepted by NFA N)

Let $N = (Q, \Sigma, \delta, q_0, F)$ be a nondeterministic finite automaton and w be a string over alphabet Σ .

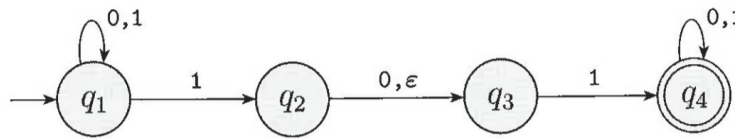
N **accepts** w if we can write w as $w = y_1 y_2 \dots y_m$, $y_i \in \Sigma_\epsilon$ and if there exists a sequence of states r_0, r_1, \dots, r_m (in Q), such that all following three conditions hold:

1. $r_0 = q_0$ (N starts in start state.)
2. $r_{i+1} \in \delta(r_i, y_{i+1})$, for $i = 0, \dots, m-1$
(N state change follows transition function.)
3. $r_m \in F$ (N ends up in accept state)

If N does not accept w , it **rejects** it.

A *computation branch* of NFA on a string is a sequence of states such that it starts in a start state and uses only transitions of δ .

A computation branch is *accepting* if the last state after all transitions is an element of F ; otherwise, it is a *rejecting* branch.

Example of NFA

Given the definition of an NFA as 5-tuple, N_1 is defined by

$$N_1 = (Q, \Sigma, \delta, q_1, F), \quad Q = \{q_1, q_2, q_3, q_4\}, \quad \Sigma = \{0, 1\}, \quad F = \{q_4\},$$

with transition function

δ	0	1	ϵ
q_1	$\{q_1\}$	$\{q_1, q_2\}$	\emptyset
q_2	$\{q_3\}$	\emptyset	$\{q_3\}$
q_3	\emptyset	$\{q_4\}$	\emptyset
q_4	$\{q_4\}$	$\{q_4\}$	\emptyset

Finally, we can to the conclusion that

$$L(N_1) = \{ w \mid w \text{ contains either } 101 \text{ or } 11 \text{ as a substring} \}.$$

Let us create a set $E(R) = \{ q \in Q \mid \exists \{s_i\}_{0..m} : s_0 \in R \wedge s_m = q \wedge s_{i+1} \in \delta(s_i, \epsilon) \}$.

A set R is a state for FA: $R \in Q'$.

A set $E(R)$ is a set of states for NFA: $E(R) \in Q$. It consists of such states that can be reached from R only via ϵ -transitions.

It is obvious that $R \in E(R)$: $\delta(r, \epsilon) = r, \forall r \in R$. Thus, we can modify our function δ' :
 $\delta'(R, a) = \bigcup_{r \in R} E(\delta(r, a)), \forall R \in Q', a \in \Sigma$. As we can see, a set of values for δ' can become only larger.

Also, we need to modify the start state: $q_0 = E(\{q_0\})$.

Corollary 2.1 (relation between RL and NFA)

Language is regular \Leftrightarrow some NFA recognizes it.

Proof:

Language is regular $\Leftrightarrow \exists$ FA that recognizes it $\Leftrightarrow \exists$ NFA that recognizes it.
Th 2.1

Th 2.2 (star operation of RLs)

The class of regular languages is closed under the star operation:

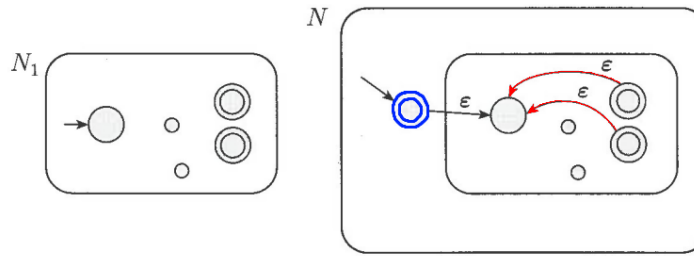
A is a RL $\Rightarrow A^*$ is an RL.

Proof:

Remainder: $A^* = \{ x_1 x_2 \dots x_k : k \geq 0 \wedge x_i \in A, \forall i \}$.

We have an NFA $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ that recognizes A . We want to construct an NFA $N = (Q, \Sigma, \delta, q_0, F)$ such that it recognizes A^* .

In the STD we **connect final states with the start state** via ϵ -transitions, thus we get a concatenation of elements. Also, there is also an empty string ϵ as an input in A^* , so we choose a new start state by adding **a new state** to the initial start state:



- $Q = \{q_0\} \cup Q_1$
- $F = \{q_0\} \cup F_1$
- $\forall q \in Q, a \in \Sigma_\epsilon :$

$$\delta(q, a) = \begin{cases} \delta_1(q, a) & \text{if } q \in Q_1 \text{ and } q \notin F_1, & (\text{main part of } N_1) \\ \delta_1(q, a) & \text{if } q \in F_1 \text{ and } a \neq \epsilon, \\ \delta_1(q, a) \cup \{q_1\} & \text{if } q \in F_1 \text{ and } a = \epsilon, & (\text{loop to old start state}) \\ \{q_1\} & \text{if } q = q_0 \text{ and } a = \epsilon, & (\text{adding } q_0) \\ \emptyset & \text{if } q = q_0 \text{ and } a \neq \epsilon, \text{ and} \end{cases}$$

23-02-20

Regular expressions

Definition 1.11 Let Σ be an alphabet. A **regular expression** (RE) R over the alphabet Σ describes a language $L(R)$ over Σ . It is inductively defined such that R is RE if R is

1. a for some $a \in \Sigma$, with $L(a) = \{a\}$
2. ϵ , with $L(\epsilon) = \{\epsilon\}$
3. \emptyset , with $L(\emptyset) = \emptyset$
4. $(R_1 \cup R_2)$ (R_1, R_2 REs), with $L(R_1 \cup R_2) = L(R_1) \cup L(R_2)$
5. $(R_1 \circ R_2)$ (R_1, R_2 REs), with $L(R_1 \circ R_2) = L(R_1) \circ L(R_2)$
6. (R_1^*) , (R_1 RE), with $L(R_1^*) = L(R_1)^*$

Parentheses in an RE can be omitted. Then, evaluation is done in the precedence order $*$, \circ , \cup . If clear from the context, the concatenation operator \circ does not have to be written down. Moreover, “ $R_1 = R_2$ ” means $L(R_1) = L(R_2)$.

$$R \cup \emptyset = R$$

$$R \circ \epsilon = R$$

$$L((\Sigma\Sigma)^*) = \{w \mid w \text{ is a string of even (and 0) length}\}.$$

Remark (Warning about common mistakes) A common mistake of beginners in the field is to e.g. write

$$0^*10^* = \{w \in \Sigma^* | w \text{ has exactly a single } 1\}$$

to indicate the language described by 0^*10^* . This statement is *wrong*, as the object on the left-hand side of the equation is a regular expression, which is *not* a language. Only $L(0^*10^*)$ on the left-hand side is correct.

Lemma 3.1

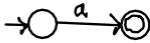

If a language is described by a regular expression, then it is regular.


Proof:

We need to build an NFA N which recognizes $L(R)$: $L(N) = L(R)$.

A regular expression consists of six elementary cases. If we are able to build an NFA for each case, then we will show that construction of the required NFA is possible.

The cases are:

- $\forall a \in \Sigma : a$ is accepted by some NFA: 
- ϵ is accepted by some NFA: 
- \emptyset is accepted by some NFA \Leftrightarrow Some NFA recognizes an empty language.

The NFA is: 

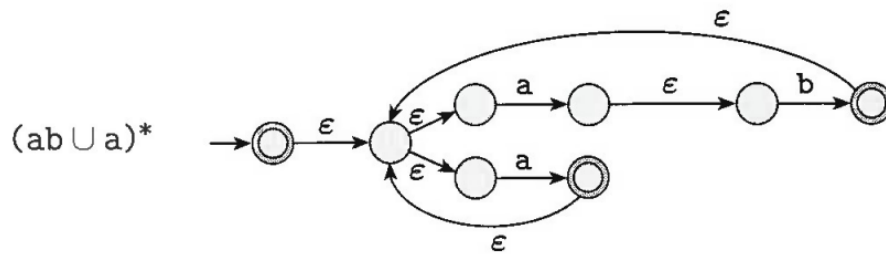
- $R_1 \cup R_2$ is accepted by some NFA. It is true, since the class of regular languages is closed under the union operation ([Th. 1.1](#))
- The same is for $R_1 \circ R_2$ and R^* ([Th. 1.2](#) and [Th. 2.2](#))

We have an NFA \Rightarrow we can build an FA ([Th. 2.1](#)) \Rightarrow the given language is regular.

Example: building an NFA

$R = (ab \cup a)^*$. Build an NFA that recognizes $L(R)$.

Build separate branches for *blue* pieces, then connect finish points with "pseudo-start" point to implement $*$.



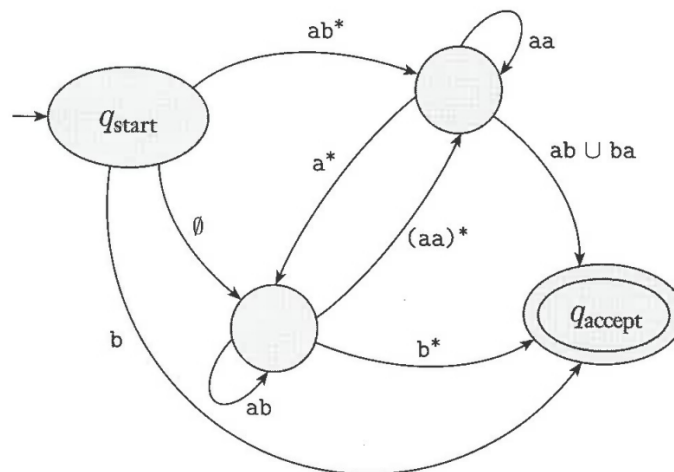
GNFA: generalized nondeterministic finite automaton

Definition

Definition 1.12 A generalized nondeterministic finite automaton (GNFA), $(Q, \Sigma, \delta, q_{start}, q_{accept})$, is a 5-tuple, with

1. Q is the finite set of states,
2. Σ is the finite input alphabet,
3. $\delta : Q \times Q \rightarrow \mathcal{R}$ (\mathcal{R} : set of all REs over Σ), where $\delta(q, q_{start})$ and $\delta(q_{accept}, q)$ are undefined for all $q \in Q$.
4. $q_{start} \in Q \setminus \{q_{accept}\}$ is the start state, and
5. $q_{accept} \in Q \setminus \{q_{start}\}$ is the accept state.

Example



A GNFA is an NFA, such that we associate to each transition a *regular expression* instead of a symbol of the alphabet.

The start state and the accept state are distinct. We require that each state, which is not the start or the accept state, has transitions to all other states, including itself. For the start state, there exist outgoing transitions to all other states, but no incoming connections. The accept state has only incoming connections that come from all states.

Definition 1.13 A GNFA $G = (Q, \Sigma, \delta, q_{start}, q_{accept})$ **accepts** a string $w \in \Sigma^*$, if there exists a decomposition $w = w_1 w_2 \cdots w_k$ ($w_i \in \Sigma^*$) and a sequence of states q_0, q_1, \dots, q_k such that

1. $q_0 = q_{start}$ is the start state
2. $q_k = q_{accept}$ is the accept state, and
3. for each $i = 1, \dots, k$ we have $w_i \in L(R_i)$, where $R_i = \delta(q_{i-1}, q_i)$.

Otherwise w is **rejected**. $L(G)$ is the **language recognized by G** .

Th. 3.1 (equivalence between FA and GNFA)

For each FA M , there is a GNFA G , such that $L(G) = L(M)$.

Proof:

We have an FA $M = (Q, \Sigma, \delta, q_0, F)$. We want to construct an GNFA $G = (Q', \Sigma, \delta', q_{start}, q_{accept})$ such that $L(G) = L(M)$.

We add states q_{start} and q_{accept} . $Q' = Q \cup \{q_{start}, q_{accept}\}$

q_{start} has an ϵ -transition to q_0 .

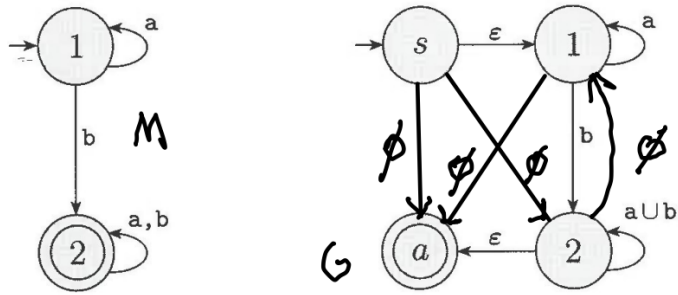
q_{accept} gets ϵ -transitions from all $q \in F$.

$$\delta'(q_1, q_2) = \begin{cases} \epsilon & \text{if } q_1 = q_{start}, q_2 = q_0 \\ \epsilon & \text{if } q_1 \in F, q_2 = q_{accept} \\ \mathcal{U}(\{a \in \Sigma \mid \delta(q_1, a) = q_2\}) & \text{if } q_1 \in Q, q_2 \in Q \\ \emptyset & \text{for all other to be defined } (q_1, q_2) \end{cases}$$

$\mathcal{U}(\mathcal{A})$ is the operation that builds a regular expression from the language \mathcal{A} . For example, $\mathcal{U}(\{a, b, c\}) = a \cup b \cup c$.

Example

FA is on the left side, GNFA is on the right side.



Lemma 3.3 \todo

If a language is regular, then it is described by a regular expression.

Proof:

\todo

Th.

Language is regular \Leftrightarrow it can be described with a regular expression.

Proof: it is true because of Lemmas 3.1 and 3.3.

23-02-27

Nonregular Languages

The *Pumping Lemma* helps to decide whether a language is not regular. Its point is that, for a regular language, strings beyond a certain size contain parts that can be repeated ("pumped up") arbitrarily often while keeping the string in the same language.

the Pumping Lemma

Σ is an alphabet and A is a language over Σ .

A is regular \Rightarrow there exists p such that any string $s \in A$ of length at least p can be divided into three pieces x, y, z which satisfy the following conditions:

- $\forall i \geq 0 : xy^iz \in A$
- $|y| > 0$
- $|xy| \leq p$

p is called *the pumping length*.

Example 1

$B = \{0^n 1^n \mid n \geq 0\}$ is a nonregular language.

Proof:

If B is regular, then $\exists p$ from the Pumping Lemma. Let us consider a string $0^p 1^p \in B$ and try to divide it into three pieces

1. y consists only of 0s. Then a string $xyyz$ has more 0s than 1s $\Rightarrow xyyz \notin B$.
2. y consists only of 1s. The same as in point 1.
3. y consists of 0s and 1s. Then in a string $xyyz$ we can find 0 and 1 that go in the wrong order (1 is before 0) $\Rightarrow xyyz \notin B$.

Example 2 \TODO

$F = \{ ww \mid w \in \{0, 1\}^* \}$ is nonregular.

Proof: \TODO

Context-Free Grammars

Like regular expressions, they are a means to describe languages.

The language class counterpart for context-free grammars will be context-free languages (CFL), similar to regular languages that are the language class counterpart for regular expressions.

Note that a CFG can be ambiguous, i.e. there might be more than one way to generate a given string.

Definition 2.1 A **context-free grammar** (CFG) is a 4-tuple (V, Σ, R, S) , where

1. V is a finite set called the **variables**,
2. Σ is a finite set, disjoint from V , called the **terminals**,
3. R is a finite set of (substitution) **rules** / **productions** of the form

$$A \rightarrow w ,$$

with $A \in V$ and $w \in (V \cup \Sigma)^*$ (i.e. $w = \varepsilon$ is possible), and

4. $S \in V$ is the **start variable**.