Automata and Processes (2IJ55)

Teacher:
H. Zantema
Room HG 6.73
tel 040 - 2472749
email: h.zantema@tue.nl

lectures — instructions — examination

Information:
www.win.tue.nl/~hzantema/ap.html

Automata part from
P. Linz
An introduction to formal languages and automata
3d edition, Jones and Bartlett ISBN 0-7637-1422-4

Process theory part from
C.A. Middelburg and M.A Reniers
Introduction to process theory

2

Ingredients of a machine / automaton:

- **(internal) states** describing the state of the machine

  There are finitely many such states; write Q for the set of all these states

- Input symbols / actions

  Again finitely many; the set Σ of all input symbols is called the **input alphabet**

- State transitions

  At every state and every input symbol there is a state where to arrive from the original state reading the input symbol or doing the corresponding action

  Mathematically described by a

  transition function $\delta : Q \times \Sigma \to Q$

- A special state $q_0 \in Q$ to start in: the **initial state**

- A set $F \subseteq Q$ of **final states**

3

These five ingredients $(Q, \Sigma, \delta, q_0, F)$ together are called a **dfa**:

deterministic finite accepter

or

deterministic finite automaton

The way we draw a dfa is called a **transition graph**, with

- **nodes (vertices)** for the states, and
- **arrows** for the transitions, each having a label from Σ

The node corresponding to q_0 is the **initial node**, drawn by an extra incoming arrow

Nodes corresponding to states in F are **final nodes**, drawn by a double circle

---

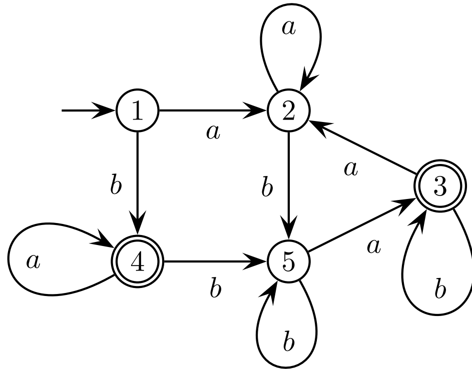
4

Example:

The dfa $(Q, \Sigma, \delta, q_0, F)$ given by

- $Q = \{1, 2, 3, 4, 5\}$, $\Sigma = \{a, b\}$,
- $\delta(1, a) = 2$, $\delta(1, b) = 4$, $\delta(2, a) = 2$, $\delta(2, b) = 5$, $\delta(3, a) = 2$, $\delta(3, b) = 3$, $\delta(4, a) = 4$, $\delta(4, b) = 5$, $\delta(5, a) = 3$, $\delta(5, b) = 5$,
- $q_0 = 1$, $F = \{3, 4\}$

is represented by the transition graph



5

Such an automaton is also called a **transition system**

Two main applications of automata / transition systems:

- Describing behaviour of systems or processes, where the states are the states of the system, and the Σ symbols represent **actions**

  This is the topic of the second part of the course:

  process theory

- Describing (formal) languages

  This is the topic of the first part of the course:

  automata theory

6

(Formal) Languages

A language is a set of sentences; each sentence may have a meaning

Every sentence is a finite sequence of symbols from the alphabet Σ

For a language one may consider

- its **syntax**: the shape, which sentence is in the language and which not
- its **semantics**: the meaning of the sentences

We will mainly concentrate on syntax

So:

A **(formal) language** is a set of sentences

7

Described mathematically:

A (formal) language is a subset of Σ^* for some finite alphabet Σ

Here Σ^* is defined to be the set of finite sequences or **strings** consisting of symbols from Σ

Example:

$\Sigma = \{a, b\}$

$\{ab, aab, bbaaabb\}$ is a finite language

$\{aba, abba, abbba, abbbba, \ldots\} =$
$\{ab^n a \mid n \geq 1\}$ is an infinite language

The first part of this course is mainly about describing (infinite) languages precisely: programming languages, specification languages, ...

8

Strings

Concatenation:

If $v = a_1 a_2 \cdots a_n$ and $w = b_1 b_2 \cdots b_m$ then we write

$$vw = a_1 a_2 \cdots a_n b_1 b_2 \cdots b_m$$

For $b \in \Sigma$ we define inductively

$b^1 = b$ and

$b^{k+1} = b^k b$ for $k \geq 1$

If $v = a_1 a_2 \cdots a_n$ then $|v| = n$ is the **length** of v

We always have: $|vw| = |v| + |w|$

9

We consider the **empty string** ϵ (pronounce: epsilon) of length 0

For every string v we have $v\epsilon = \epsilon v = v$, so ϵ is a **neutral element** for string concatenation, just like 0 is a neutral element for addition and 1 for multiplication

For every symbol a we define

$$a^0 = \epsilon$$

If $v = a_1 a_2 \cdots a_n$ then $v^R = a_n a_{n-1} \cdots a_1$ is called the **reverse** of v

We always have: $(v^R)^R = v$

10

A consecutive part of a string is called a **substring**, more precisely, v is a substring of w if

$$w = uvu'$$

for strings u, u'

If $w = uv$ then u is called a **prefix** of w, and v a **suffix** of w

Since we defined a language to be a set of strings, all set theoretic notions like $\in$, $\subseteq$, $\cap$, $\cup$, $-$, $\ldots$ are defined and may be used for languages

Now we we define concatenation and reverse also for languages rather than strings:

$$L_1 L_2 = \{xy \mid x \in L_1 \wedge y \in L_2\}$$

$$L^R = \{x^R \mid x \in L\}$$

11

The **complement** $\overline{L}$ of a language L is defined by

$$\overline{L} = \Sigma^* - L$$

i.e., the set of all strings that are not in L

We define

$L^0 = \{\epsilon\}$

$L^1 = L$

$L^{n+1} = L^n L$ for every $n \geq 1$

$$L^* = \bigcup_{i=0}^{\infty} L^i = L^0 \cup L^1 \cup L^2 \cup L^3 \cup \cdots$$

$$L^+ = \bigcup_{i=1}^{\infty} L^i = L^1 \cup L^2 \cup L^3 \cup \cdots$$

12

Example:

If $L = \{ab^n \mid n \geq 0\} = \{a, ab, abb, \ldots\}$, then

$L^2 = \{ab^n ab^m \mid m, n \geq 0\}$

Note that generally speaking

$$L^2 = \{vw \mid v, w \in L\} \neq \{vv \mid v \in L\}$$

For the same L one can prove

$L^+ = \{a\}\{a, b\}^*$, that is, the set of all non-empty strings starting by a

Such a proof of an equality of the shape $A = B$ is given as we know from set theory:

- Choose $x \in A$ arbitrarily, prove that $x \in B$
- Choose $x \in B$ arbitrarily, prove that $x \in A$

13

Grammars

Grammars provide a way to define languages

Example:

a programming language consisting of statements, variables, expressions, etc. may be described as follows:

$$
\begin{array}{rcl}
\langle\text{stm}\rangle & \rightarrow & \langle\text{var}\rangle := \langle\text{expr}\rangle \\
\langle\text{stm}\rangle & \rightarrow & \langle\text{stm}\rangle\ ;\ \langle\text{stm}\rangle \\
\langle\text{var}\rangle & \rightarrow & x \\
\langle\text{var}\rangle & \rightarrow & y \\
\langle\text{expr}\rangle & \rightarrow & 2 \\
\langle\text{expr}\rangle & \rightarrow & 3 \\
\cdots & \rightarrow & \cdots
\end{array}
$$

Now you can compose the statement

$$x := 2;\ y := 3$$

$\langle\text{stm}\rangle$

$\Rightarrow\ \langle\text{stm}\rangle\ ;\ \langle\text{stm}\rangle$

$\Rightarrow\ \langle\text{var}\rangle := \langle\text{expr}\rangle\ ;\ \langle\text{stm}\rangle$

$\Rightarrow x := \langle\text{expr}\rangle\ ;\ \langle\text{stm}\rangle$

$\Rightarrow x := 2;\ \langle\text{stm}\rangle$

$\Rightarrow x := 2;\ \langle\text{var}\rangle := \langle\text{expr}\rangle$

$\Rightarrow x := 2;\ y := \langle\text{expr}\rangle$

$\Rightarrow x := 2;\ y := 3$

A set of such rules is called a **context free grammar**, or shortly **grammar**

More precisely:

A (context free) grammar $G = (V, T, S, P)$ consists of

- a finite set V of **variables** or **non-terminals**

  (in this example: $\langle\text{stm}\rangle$, $\langle\text{var}\rangle$, $\langle\text{expr}\rangle$)

- a finite set T of **terminal symbols** or shortly terminals

  (in this example: ";", x, y, ":=", 2, 3)

- a special symbol $S \in V$, the **start variable**

  (in this example: $\langle\text{stm}\rangle$)

- a finite set P of **productions** $u \rightarrow v$ in which
  - $u \in V$ and
  - $v \in (V \cup T)^*$

A **derivation step** on a string is: replace a substring u by v for some production $u \rightarrow v$

More precisely: if $u \rightarrow v$ is a production, then

$$wux \Rightarrow wvx$$

is a derivation step for any $w, x \in (V \cup T)^*$

A **derivation** $u \Rightarrow^* v$ from u to v means that v is obtained from u by applying 0 or more derivation steps

More precisely: there are $w_1, w_2, \ldots, w_n$ such that

$$u = w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n = v$$

Definition:

For a grammar $G = (V, T, S, P)$

$$L(G) = \{w \in T^* \mid S \Rightarrow^* w\}$$

is the **language generated by** G

Example:

$$G = (\{S\}, \{a, b\}, S, P)$$

where P consists of

$$S \to aSb$$

$$S \to \epsilon$$

yields the derivation

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb$$

hence $S \Rightarrow^* aaabbb$ and $aaabbb \in T^*$
so $aaabbb \in L(G)$

By applying the first production n times, followed by applying once the second, one obtains $a^n b^n \in L(G)$ for every $n \geq 0$

Conversely there is no other way to construct a derivation $S \Rightarrow^* w$ satisfying $w \in T^*$, hence

$$L(G) = \{a^n b^n \mid n \geq 0\}$$

18

In defining $G = (V, T, S, P)$ we often use capitals for variables and lower case letters for terminals

Usually we only give productions, and leave implicit that

- V is the set of capital symbols occurring in the productions
- T is the set of lower case symbols occurring in the productions
- S is the start symbol

Example:

G_1 :			G_2 :		
S	$\to$	Ab	S	$\to$	aSb
A	$\to$	aAb	S	$\to$	b
A	$\to$	ϵ			

Now we have $L(G_1) = L(G_2) = \{a^n b^{n+1} \mid n \geq 0\}$

Two grammars G_1 and G_2 are called **equivalent** if $L(G_1) = L(G_2)$

19

Alternative convention:

- put variable names inside $\langle$ and $\rangle$
- all other words are terminals

Example:

$$\begin{array}{rcl}
\langle \text{stm} \rangle & \to & \langle \text{var} \rangle \ := \ \langle \text{expr} \rangle \\
\langle \text{stm} \rangle & \to & \langle \text{stm} \rangle \ ; \ \langle \text{stm} \rangle \\
\langle \text{stm} \rangle & \to & \text{begin} \ \langle \text{stm} \rangle \ \text{end} \\
\langle \text{stm} \rangle & \to & \text{if} \ \langle \text{cond} \rangle \ \text{then} \ \langle \text{stm} \rangle \ \text{else} \ \langle \text{stm} \rangle \\
\langle \text{stm} \rangle & \to & \text{if} \ \langle \text{cond} \rangle \ \text{then} \ \langle \text{stm} \rangle \\
\langle \text{stm} \rangle & \to & \text{while} \ \langle \text{cond} \rangle \ \text{do} \ \langle \text{stm} \rangle \\
\langle \text{cond} \rangle & \to & \cdots \\
\langle \text{var} \rangle & \to & \cdots \\
\langle \text{expr} \rangle & \to & \cdots \\
\cdots & \to & \cdots
\end{array}$$

B(ackus) N(aur) F(orm)

write $A \ ::= \ u \mid v \mid w$ rather than

$$A \to u, \quad A \to v, \quad A \to w$$

20

Example:

$$G: \quad \begin{array}{rcl}
S & \to & SS \\
S & \to & \epsilon \\
S & \to & aSb \\
S & \to & bSa
\end{array}$$

can be written shortly as

$$S \ ::= \ SS \mid \epsilon \mid aSb \mid bSa$$

This grammar generates exactly all strings in which the number of a's is equal to the number of b's

More precisely: $L(G) = \{w \mid n_a(w) = n_b(w)\}$

where n_a, n_b denote the number of a's and. b's, respectively

This can be proved formally using induction

21

Back to automata $\cdots$

Just like a grammar also a dfa defines a language:

If $v = a_1 a_2 \cdots a_n \in \Sigma^*$ then you can follow a path in the transition graph starting in q_0 and consecutively following the arrows labeled by $a_1, a_2, \ldots, a_n$

In case the last node reached is final then v is in the language, otherwise it is not

---

22

More precisely, $\delta : Q \times \Sigma \to Q$ is extended to

$$\delta^* : Q \times \Sigma^* \to Q,$$

defined inductively by

$$\delta^*(q, \epsilon) = q$$
$$\delta^*(q, wa) = \delta(\delta^*(q, w), a)$$

Indeed this is an extension, since

$$\delta^*(q, a) = \delta^*(q, \epsilon a) = \delta(\delta^*(q, \epsilon), a) = \delta(q, a)$$

For a dfa

$$M = (Q, \Sigma, \delta, q_0, F)$$

the corresponding language is defined by

$$L(M) = \{w \in \Sigma^* \mid \delta^*(q_0, w) \in F\}$$

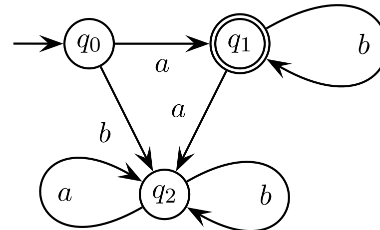
This is called the **language accepted by** M

23

Example:

$M = (Q, \Sigma, \delta, q_0, F)$, where

- $Q = \{q_0, q_1, q_2\}$,
- $\Sigma = \{a, b\}$
- $\delta(q_0, a) = q_1$, $\delta(q_0, b) = q_2$, $\delta(q_1, a) = q_2$, $\delta(q_1, b) = q_1$, $\delta(q_2, a) = q_2$, $\delta(q_2, b) = q_2$
- $F = \{q_1\}$



Then $L(M) = \{ab^n \mid n \geq 0\}$

Such a state q_2 from which never a final state can be reached is called a **trap state**

24

Definition:

A language L is called **regular** if a dfa M exists satisfying $L(M) = L$

We have seen that $\{ab^n \mid n \geq 0\}$ is a regular language

Later we will see that $\{a^n b^n \mid n \geq 0\}$ is not a regular language

Theorem

If L is a regular language then $\overline{L}$ is a regular language too

Sketch of proof:

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a dfa satisfying $L(M) = L$

Then $M' = (Q, \Sigma, \delta, q_0, Q - F)$ is a dfa satisfying $L(M') = \overline{L}$

25

Until now for every string there is exactly one path through the automaton starting from the initial state:

the automaton is **deterministic**

This is because for every state $q \in Q$ and every $a \in \Sigma$ there is exactly one arrow starting in q labeled by a: δ is a function from $Q \times \Sigma$ to Q

Now we want to allow that there are no such arrows, or more than one

We also want to allow **empty steps**: steps allowed to be done in the automaton without eating a symbol from the string

---

26

The resulting type of automaton is called an **nfa:**

non-deterministic finite accepter

The definition coincides with the definition of dfa except for the type of δ: now we have

$$\delta : Q \times (\Sigma \cup \{\epsilon\}) \to \mathcal{P}(Q)$$

Here $\mathcal{P}(Q)$ is the **power set** of Q: the set of all subsets of Q

The transition graph notation is as follows:

> For $q, r \in Q$ and $a \in \Sigma \cup \{\epsilon\}$ there is an arrow from q to r labeled by a if and only if $r \in \delta(q, a)$

27

Example:

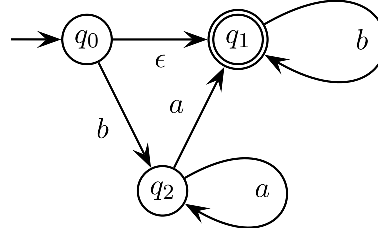
$M = (Q, \Sigma, \delta, q_0, F)$, where

- $Q = \{q_0, q_1, q_2\}$,
- $\Sigma = \{a, b\}$
- $\delta(q_0, a) = \emptyset$, $\delta(q_0, b) = \{q_2\}$, $\delta(q_0, \epsilon) = \{q_1\}$,

  $\delta(q_1, a) = \emptyset$, $\delta(q_1, b) = \{q_1\}$, $\delta(q_1, \epsilon) = \emptyset$,

  $\delta(q_2, a) = \{q_1, q_2\}$, $\delta(q_2, b) = \emptyset$, $\delta(q_2, \epsilon) = \emptyset$,
- $F = \{q_1\}$

is represented by the transition graph



28

For dfa's we defined

$$L(M) = \{w \in \Sigma^* \mid \delta^*(q_0, w) \in F\}$$

For nfa's we define

$$L(M) = \{w \in \Sigma^* \mid \delta^*(q_0, w) \cap F \neq \emptyset\}$$

Therefore first we need to define

$$\delta^* : Q \times \Sigma^* \to 2^Q$$

which is done recursively:

$$\delta^*(q, \epsilon) = \bigcup_{i=0}^{\infty} A_i,$$

where

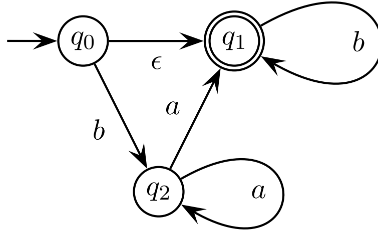
$$A_0 = \{q\}, \quad A_{i+1} = \bigcup_{r \in A_i} \delta(r, \epsilon)$$

$$\delta^*(q, wa) \;=\; \{r \in Q \mid \exists s, s' \in Q :$$
$$s \in \delta^*(q, w) \wedge s' \in \delta(s, a) \wedge r \in \delta^*(s', \epsilon)\}$$

29

Intuitively: $r \in \delta^*(q, w)$ means that there is a path in the transition graph from q to r, labeled by the string w, while this path may contain empty steps that are ignored in building the path

For example, in



we have

$$L(M) = \{b^k \mid k \geq 0\} \cup \{ba^n b^k \mid n > 0,\ k \geq 0\}$$

30

The definition of $L(M)$ for an nfa M is slightly more complicated than for a dfa M

But the definition of an nfa gives much more freedom: we do not have any more the requirement that for every q and a there is exactly one outgoing arrow from q labeled by a, but every configuration is allowed

For instance, for given dfa's or nfa's M_1, M_2 it is easy to construct an nfa M satisfying

$$L(M) = L(M_1) \cup L(M_2)$$

while constructing a dfa M with the same property is much harder

Is the formalism of nfa's more powerful than the formalism of dfa's?

More precisely, are there languages that are accepted by an nfa, but not by a dfa?

NO

31

Theorem

A language is accepted by an nfa if and only it is accepted by a dfa

Proof sketch:

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a dfa

Then $M' = (Q, \Sigma, \delta', q_0, F)$ defined by

$$\delta'(q, a) = \{\delta(q, a)\}$$

$$\delta'(q, \epsilon) = \emptyset$$

is an nfa satisfying $L(M') = L(M)$: the nfa having the same transition graph

Conversely, let $M = (Q, \Sigma, \delta, q_0, F)$ be an nfa

We will construct a dfa

$$M_D = (Q_D, \Sigma, \delta_D, q_{0D}, F_D)$$

satisfying $L(M_D) = L(M)$, then we are done

32

$$Q_D = \mathcal{P}(Q)$$

$$\delta_D(X, a) = \bigcup_{q \in X} \delta^*(q, a)$$

$$q_{0D} = \delta^*(q_0, \epsilon)$$

$$F_D = \{X \subseteq Q \mid X \cap F \neq \emptyset\}$$

For every $w \in \Sigma^*$ and $q \in Q$ one can prove

$$q \in \delta^*(q_0, w) \iff q \in \delta_D^*(q_{0D}, w)$$

As a consequence one obtains $L(M_D) = L(M)$

End of proof sketch

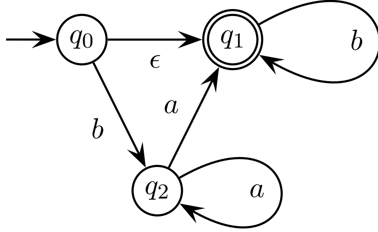
33

In order to use this for a construction one may restrict to the reachable part of $\mathcal{P}(Q)$, i.e., only introduce new states of the shape

$$\delta_D(X, a) = \bigcup_{q \in X} \delta^*(q, a)$$

starting from $q_{0D} = \delta^*(q_0, \epsilon)$

As an example, we do this construction for our nfa



We start by the initial node $\delta^*(q_0, \epsilon) = \{q_0, q_1\}$

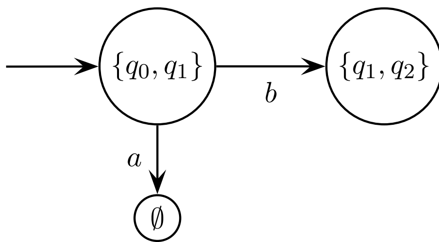
34

Next we compute

$$\delta_D(\{q_0, q_1\}, a) = \bigcup_{q \in \{q_0, q_1\}} \delta^*(q, a)$$

and

$$\delta_D(\{q_0, q_1\}, b) = \bigcup_{q \in \{q_0, q_1\}} \delta^*(q, b)$$



Next we compute

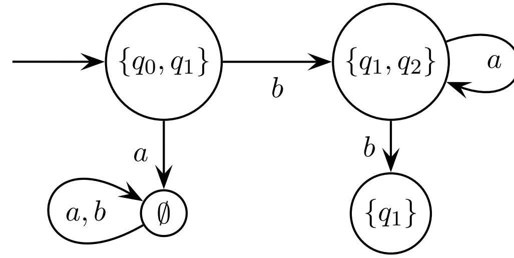
$$\delta_D(X, a) = \bigcup_{q \in X} \delta^*(q, a)$$

and

$$\delta_D(X, b) = \bigcup_{q \in X} \delta^*(q, b)$$

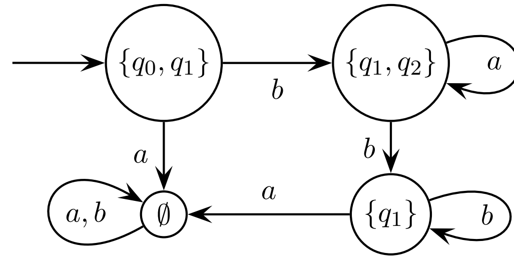
for the two new states X

35



where an arrow labeled by a, b is a short hand notation for two arrows respectively labeled by a, b

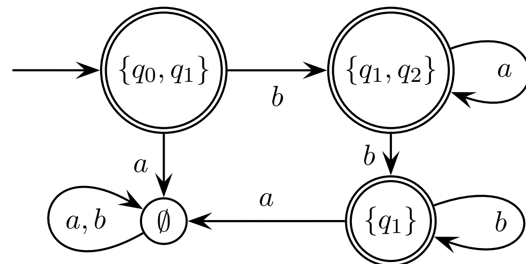
We continue by doing the same for the new state $\{q_1\}$:



36

Now no new states in the dfa, being sets of states in the original nfa, have been generated

The procedure concludes by establishing the final nodes, that are the nodes labeled by sets containing at least one original final node, in this case q_1:



Since the language accepted by the original

nfa is

$$L(M) = \{b^k \mid k \geq 0\} \cup \{ba^n b^k \mid n > 0,\ k \geq 0\}$$

the same holds for the dfa that has been constructed now

37

We have seen two equivalent characterizations of regular languages:

- a regular language is a language L satisfying

  $L = L(M)$ for some dfa M

- a regular language is a language L satisfying

  $L = L(M)$ for some nfa M

We will obtain some more characterizations, all being equivalent:

- using regular expressions

- using regular grammars

38

Regular expressions

We have seen several ways to construct new languages from existing ones:

$L_1 \cup L_2$

$L_1 L_2 = \{vw \mid v \in L_1 \wedge w \in L_2\}$

$L^* = L^0 \cup L^1 \cup L^2 \cup L^3 \cup \cdots$

Regular expressions describe languages starting by single symbols, or empty string or empty set, and using these constructions

In regular expressions the union construction is written by the symbol "+"

39

Definition

A regular expression over an alphabet Σ is a expression obtained by applying the following rules a finite number of times:

- $\emptyset$ is a regular expression

- ϵ is a regular expression

- a is a regular expression for every $a \in \Sigma$

- r^* is a regular expression for every regular expression r

- $(r_1 + r_2)$ is a regular expression for all regular expressions r_1 and r_2

- $(r_1 \cdot r_2)$ is a regular expression for all regular expressions r_1 and r_2

40

Example

$$((\epsilon + b^*) \cdot (c^* \cdot \emptyset))$$

is a regular expression over $\Sigma = \{a, b, c\}$

Just as in expressions concerning addition and multiplication we often omit redundant parentheses

We give $\cdot$ a higher priority than $+$, i.e.,

$$r_1 \cdot r_2 + r_3 = ((r_1 \cdot r_2) + r_3)$$

$$r_1 \cdot r_2 + r_3 \neq (r_1 \cdot (r_2 + r_3))$$

Sometimes we even omit the symbol $\cdot$, just as we are used to do in multiplication

---

41

Every regular expression r defines a language $L(r)$ as follows

- $L(\emptyset) = \emptyset$
- $L(\epsilon) = \{\epsilon\}$
- $L(a) = \{a\}$ for every $a \in \Sigma$
- $L(r^*) = (L(r))^*$
  for every regular expression r
- $L(r_1 + r_2) = L(r_1) \cup L(r_2)$
  for all regular expressions r_1 en r_2
- $L(r_1 \cdot r_2) = L(r_1)L(r_2)$
  for all regular expressions r_1 en r_2

42

Example

$$\begin{aligned} L((a+b)\cdot c^*) &= L(a+b)L(c^*) \\ &= (L(a) \cup L(b))L(c^*) \\ &= (\{a\} \cup \{b\})L(c^*) \\ &= \{a,b\}L(c^*) \\ &= \{a,b\}\{c\}^* \end{aligned}$$

$$= \{a, ac, acc, accc, \ldots, b, bc, bcc, bccc, \ldots\}$$

Exercise

Find a regular expression r over $\{a,b\}$ such that $L(r)$ consists exactly of all strings containing at least one pair of consecutive a's

Such a string always contains the substring aa; before it and behind it everything is allowed

'Everything': $\Sigma^* = \{a,b\}^* = L((a+b)^*)$

So:

$$r = (a+b)^*aa(a+b)^*$$

43

Such expression is called "regular" on purpose:

Theorem:

A language L is regular if and only if there is a regular expression r satisfying

$$L(r) = L$$

We will prove this theorem as follows

For every regular expression r we describe how to construct an nfa M satisfying $L(M) = L(r)$

and

For every nfa M we describe how to construct a regular expression r satisfying $L(r) = L(M)$

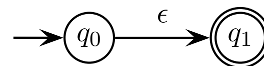
44

For every building block of a regular expression we now construct a corresponding nfa having exactly one final state accepting the right language

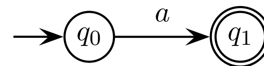
The three base cases:

The nfa for $\emptyset$:

The nfa for ϵ:

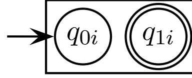


The nfa for $a \in \Sigma$:

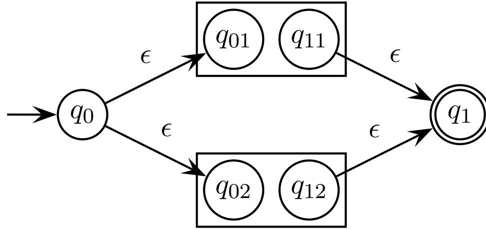


45

For the other three building blocks of regular expressions it is assumed that for r_i, $i = 1, 2$, the nfa's with one final state have been constructed already, denoted as



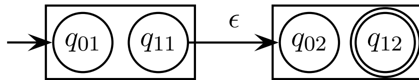
The nfa for $r_1 + r_2$:



Note: q_{11} and q_{12} are not final any more

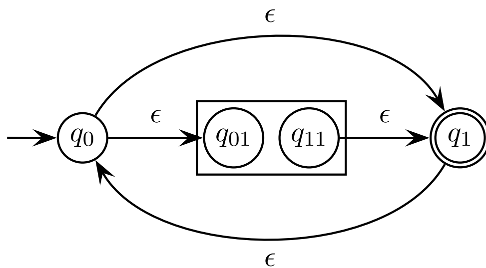
46

The nfa for $r_1 \cdot r_2$:



Here q_{11} is not final any more; q_{12} is the final state

The nfa for r_1^*:



47

Composing these constructions yields for every regular expression r a construction of an nfa M satisfying $L(M) = L(r)$

Conversely we give a construction starting from an nfa M and yielding a regular expression r satisfying $L(r) = L(M)$

In case M has more than one final state, add a new state q, add ϵ-transitions from the old final states to q and define $F = \{q\}$

This does not change the accepted language, and yields an nfa having exactly one final state

Now we will remove all nodes outside $\{q_0\} \cup F$ one by one

48

In doing so we use **generalized transition graphs**, being transition graphs in which the arrows are not labeled by ϵ or $a \in \Sigma$, but by arbitrary regular expressions over Σ

The language described by such a generalized transition graph is the set of strings that can be written as $v_1 v_2 \cdots v_n$ such that

- $v_i \in L(r_i)$ for $i = 1, \ldots, n$, and
- for which a path exists starting in q_0 and ending in the final state, and
- for which the arrows are labeled consecutively by $r_1, r_2, \ldots, r_n$

If the arrows are labeled by ϵ or $a \in \Sigma$ then this graph describes an nfa and this language coincides with the language accepted by the nfa

49

Overview of the construction:

- Start by the transition graph of the nfa having exactly one final state
- As long as a node exists outside $\{q_0\} \cup F$, remove it and add or change transitions in such a way that the described language does not change

  (In these steps the graph decreases, but the regular expressions in the labels increase in complexity)

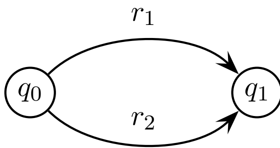
By using + one forces that for every pair q_1, q_2 of nodes there is at most one arrow from q_1 to q_2

- If there are no more such states, then the graph contains at most two nodes, and for this simple graph the desired regular expression can be given

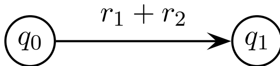
---

The details

Through the whole procedure replace



by



by which for every two nodes q_0, q_1 there are is at most one arrow from q_0 to q_1

The hardest step is node removal, and is done as follows

If q will be removed, then

- for every arrow from any node q_1 to q labeled by r_1 and
- for every arrow from q to any node q_2 labeled by r_2

a new arrow from q_1 to q_2 is added, labeled by

$$r_1 r_2$$

if there is no arrow from q to q, and labeled by

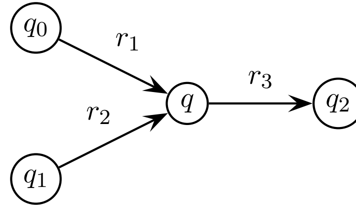
$$r_1 r^* r_2$$

if the arrow from q to q is labeled by r

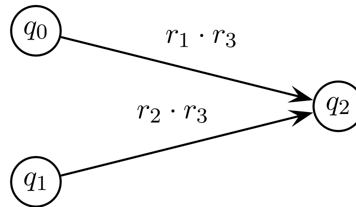
After adding these new labeled arrows (which may be many), the node q may be removed without changing the described language

---

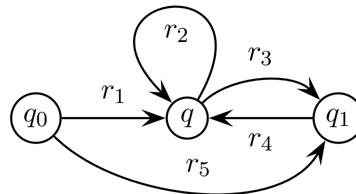
Example



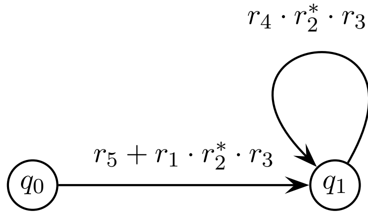
is replaced by



Example



is replaced by

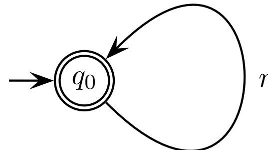


Be aware that the transition graph may blow up: if q has k incoming arrows and n outgoing arrows, then $k \times n$ new arrows are created

---

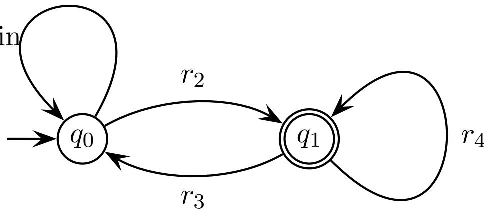
54

This procedure is continued until all non-final and non-initial nodes have been removed, resulting either in



yielding the regular expression r^*

or in



yielding the regular expression

$$r_1^* \cdot r_2 \cdot (r_4 + r_3 \cdot r_1^* \cdot r_2)^*$$

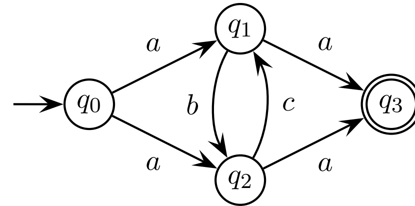
The latter may be simpler if one or more from r_1, r_2, r_3, r_4 are missing
For example, if r_3 is missing then the resulting regular expression is $r_1^* \cdot r_2 \cdot r_4^*$

55

Remark:

The order of removing nodes may influence the result

Example:



If first q_1 is removed and then q_2, then the resulting regular expression will be

$$a \cdot a + (a + a \cdot b) \cdot (c \cdot b)^* \cdot (a + c \cdot a)$$

If first q_2 is removed and then q_1, then the resulting regular expression will be

$$a \cdot a + (a + a \cdot c) \cdot (b \cdot c)^* \cdot (a + b \cdot a)$$

Although different, both regular expressions are correct: they both describe the same language as the given nfa

---

56

Regular grammars

A grammar is called **right-linear** if all productions are of the shape

$$A \to xB$$

or

$$A \to x$$

where $A, B \in V$ and $x \in T^*$

A grammar is called **left-linear** if all productions are of the shape

$$A \to Bx$$

or

$$A \to x$$

where $A, B \in V$ and $x \in T^*$

57

A grammar is called **regular** if it is either left-linear or right-linear

Note: $\epsilon \in T^*$ or a single variable may occur as the right hand side of a production, but no combination of productions of the shape $A \to Bx$ and $C \to yD$ where $x \neq \epsilon \neq y$

Theorem: A language L is regular if and only if there is a right-linear grammar G satisfying $L(G) = L$

This theorem follows from two constructions:

For a right-linear grammar G we construct an nfa M satisfying $L(M) = L(G)$

For an nfa M we construct a right-linear grammar G satisfying $L(G) = L(M)$

58

First for a right-linear grammar G we construct a corresponding nfa M

For every $A \in V$ create a state, and choose $q_0 = S$

Add a final state A_f, and choose $F = \{A_f\}$

For every production of the shape

$$A \to a_1 a_2 \cdots a_n B$$

add $n-1$ new states, and transitions

$$A \xrightarrow{a_1} \cdot \xrightarrow{a_2} \cdot \quad \cdots \quad \cdot \xrightarrow{a_n} B$$

A production $A \to B$ yields $A \xrightarrow{\epsilon} B$

59

For every production of the shape

$$A \to a_1 a_2 \cdots a_n$$

add $n-1$ new states, and transitions

$$A \xrightarrow{a_1} \cdot \xrightarrow{a_2} \cdot \quad \cdots \quad \cdot \xrightarrow{a_n} A_f$$

A production $A \to \epsilon$ yields $A \xrightarrow{\epsilon} A_f$

This construction has been designed in such a way that a derivation from S to v exists if and only if there is a path from $q_0 = S$ to A_f in this transition graph in which the consecutive labels are exactly the elements of v

This shows one direction of the theorem

60

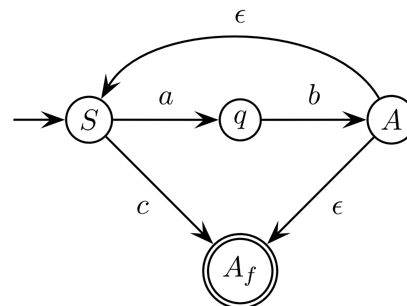
Example:

The right-linear grammar

$$S \;\to\; abA \mid c$$

$$A \;\to\; S \mid \epsilon$$

transforms to the nfa



61

Conversely we show that every regular language is generated by a right-linear grammar

For an nfa

$$M = (Q, \Sigma, \delta, q_0, F)$$

we construct a right-linear grammar

$$G = (V, \Sigma, S, P)$$

satisfying

$$L(G) = L(M)$$

Choose $V = Q$ and $S = q_0$

For every transition $r \in \delta(q, a)$ create a production

$$q \to ar$$

and for every final state $q \in F$ create a production

$$q \to \epsilon$$

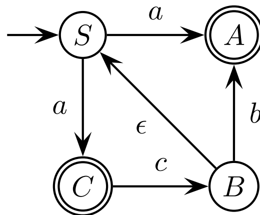
This construction has been designed in such a way that a derivation from S to v exists if and only if there is a path from $q_0 = S$ to an element of F in this transition graph in which the consecutive labels are exactly the elements of v

---

62

Example:

The nfa (in which the states already have been named S, A, B and C)



transforms to the right-linear grammar

$$S \;\to\; aA \mid aC$$

$$A \;\to \epsilon$$

$$B \;\to\; S \mid bA$$

$$C \;\to\; cB \mid \epsilon$$

63

By symmetry, we have a similar result for left-linear grammars:

Theorem: A language L is regular if and only if there is a left-linear grammar G satisfying $L(G) = L$

We conclude that the following claims are all equivalent

- L is regular
- there exists a dfa M satisfying $L(M) = L$
- there exists an nfa M satisfying $L(M) = L$
- there exists a regular expression r satisfying

  $L(r) = L$
- there exists a left-linear grammar G satisfying

  $L(G) = L$
- there exists a right-linear grammar G satisfying

  $L(G) = L$
- there exists a regular grammar G satisfying

  $L(G) = L$

64

One may think that all languages are regular ...

Claim:

$L = \{a^n b^n \mid n \geq 0\}$ is not regular

Proof:

Assume $M = (Q, \Sigma, \delta, q_0, F)$ is a dfa satisfying

$$L(M) = L$$

Q is finite, hence the states $\delta^*(q_0, a^i)$ for $i \geq 0$ are not all different

So i and j exist satisfying $i > j \geq 0$ and

$$\delta^*(q_0, a^i) = \delta^*(q_0, a^j)$$

Then we obtain $\delta^*(q_0, a^i b^j) = \delta^*(q_0, a^j b^j)$

Since $a^i b^j \notin L$ and $a^j b^j \in L$ we have

$$\delta^*(q_0, a^i b^j) \notin F \quad \text{and} \quad \delta^*(q_0, a^j b^j) \in F,$$

contradiction, so L is not regular

Conclusions language theory:

- Automata provide a natural framework to define languages formally

  Two basic types of automata: dfa's and nfa's

- Other natural frameworks to define languages formally:

  - (context free) grammars, special case: right-linear grammars
  - regular expressions

- dfa's, nfa's, regular expressions and right-linear grammars can be transformed to each other describing the same language, so they have the same expressive power

  The corresponding languages are called *regular*

- General grammars are more powerful:

  $$\{a^n b^n \mid n \geq 0\}$$

  is not regular, but can be described by a grammar