Formal Languages-Course 2.

Géraud Sénizergues

Bordeaux university

07/05/2020

Master computer-science MINF19, IEI, 2019/20

contents

Regular languages

Prolog :arithmetical expressions

Let $\oplus, \otimes$ be binary symbols, $-$ be a unary symbol. Here are arithmetical expressions, with operator-symbols $\{\oplus, \otimes, -\}$ over the alphabet of constant symbols $\Sigma = \{0, 1\}$:

$$e_1 = 0, \quad e_2 = \langle\langle\langle 1 \oplus 1\rangle \oplus 0\rangle \otimes 1\rangle, \quad e_3 = \langle\langle\langle -1\rangle \oplus 1\rangle \oplus 1\rangle$$

These are words on the alphabet $A = \{0, 1, \oplus, \times, \langle, \rangle\}$.

Prolog : arithmetical expressions

The set of all correct arithmetical expressions is the least language $\mathrm{AE} \subseteq A^*$ fulfilling :$\forall e, e' \in \mathrm{AE}$,

$$0 \in \mathrm{AE}, \quad 1 \in \mathrm{AE}$$

$$\langle e \oplus e' \rangle \in \mathrm{AE}$$

$$\langle e \otimes e' \rangle \in \mathrm{AE}$$

$$\langle -e \rangle \in \mathrm{AE}$$

$$\langle e \rangle \in \mathrm{AE}$$

Prolog :arithmetical expressions,value

The value $\nu(e)$ of an arithmetical expression e is the integer defined (inductively) by :$\forall e, e' \in \mathrm{AE}$,

$$\nu(0) = 0 \quad \nu(1) = 1 \in \mathrm{AE}$$

$$\nu(\langle e \oplus e' \rangle) = \nu(e) + \nu(e')$$

$$\nu(\langle e \otimes e' \rangle) = \nu(e) \times \nu(e')$$

$$\nu(\langle -e \rangle) = -\nu(e).$$

$$\nu(\langle e \rangle) = \nu(e)$$

Prolog :arithmetical expressions,value

$$\nu(e_1) = \nu(0) = 0.$$

$$\begin{aligned}
\nu(e_2) &= \nu(\langle\langle\langle 1 \oplus 1\rangle \oplus 0\rangle \otimes 1\rangle) \\
&= \nu(\langle\langle 1 \oplus 1\rangle \oplus 0\rangle) \cdot \nu(1) \\
&= (\nu(\langle 1 \oplus 1\rangle) + \nu(0)) \cdot \nu(1) \\
&= ((\nu(1) + \nu(1)) + 0) \cdot 1 \\
&= ((1 + 1) + 0) \cdot 1 \\
&= 2.
\end{aligned}$$

Prolog :arithmetical expressions,value

$$\begin{aligned}
\nu(e_3) &= \nu(\langle\langle\langle -1\rangle \oplus 1\rangle \oplus 1\rangle) \\
&= \nu(\langle\langle -1\rangle \oplus 1\rangle) + \nu(1) \\
&= (\nu(\langle -1\rangle) + \nu(1)) + 1 \\
&= ((-1) + 1) + 1 \\
&= 0 + 1 \\
&= 1
\end{aligned}$$

Remark : the above rewritings are by no means an algorithm for computing $\nu(*)$; they are just illustrating why the previous inductive properties of $\nu(*)$ really define $\nu(*)$.

Regular expressions :example

Let $\oplus, \otimes$ be binary symbols, $\star$ be a unary symbol,0 be a nullary symbol. Here are regular expressions, with operator-symbols $\{\oplus, \otimes, \star\}$ over the alphabet of constant symbols $\Sigma = \{a, b, c\}$:

$$e_1 = a, \quad e_2 = \langle\langle a \oplus b\rangle + c\rangle \otimes a, \quad e_3 = \langle\langle\langle a \oplus b\rangle \otimes a\rangle \star \rangle,$$

These are words on the alphabet $A = \Sigma \cup \{0, \oplus, \otimes, \star, \langle, \rangle\}$.

Regular expressions :definition

Let Σ be an alphabet. Let $\oplus, \otimes$ be binary symbols, $\star$ be a unary symbol, 0 be a nullary symbol. Let $A = \Sigma \cup \{\oplus, \times, \star, 0, \langle, \rangle\}$ The set of all correct regular expressions over Σ is the least language $\mathrm{RE} \subseteq A^*$ fulfilling :$\forall x \in \Sigma, \forall e, e' \in \mathrm{AE}$,

$$0 \in \mathrm{RE}, x \in \mathrm{RE}$$

$$\langle e \oplus e' \rangle \in \mathrm{RE}$$

$$\langle e \otimes e' \rangle \in \mathrm{RE}$$

$$\langle e \star \rangle \in \mathrm{RE}$$

$$\langle e \rangle \in \mathrm{RE}$$

Regular expressions :value

The value $\nu(e)$ of a regular expression e is the language defined (inductively) by :$\forall x \in \Sigma, e, e' \in \mathrm{AE}$,

$$\nu(0) = \emptyset \quad \nu(x) = \{x\}$$

$$\nu(\langle e \oplus e' \rangle) = \nu(e) \cup \nu(e')$$

$$\nu(\langle e \otimes e' \rangle) = \nu(e) \times \nu(e')$$

$$\nu(\langle e \star \rangle) = \nu(e)^*.$$

$$\nu(\langle e \rangle) = \nu(e)$$

Regular expressions : value

$$e_1 = a, \quad e_2 = \langle\langle a \oplus b\rangle \oplus c\rangle \otimes a, \quad e_3 = \langle\langle\langle a \oplus b\rangle \otimes a\rangle \star \rangle,$$

$$\nu(a) = \{a\},$$

$$\nu(\langle\langle a \oplus b\rangle \oplus c\rangle \otimes a) = \{aa, ba, ca\}$$

$$\nu(\langle\langle\langle a \oplus b\rangle \otimes a\rangle \star \rangle) = \{aa, ba\}^*$$

$$\nu(\langle\langle\langle a \oplus b\rangle \otimes a\rangle \star \rangle) = \{\varepsilon, aa, ba, aaaa, aaba, baaa, baba,\} \cup \{aaaaaa, aaaaba, aabaaa, aababa, baaaaa, baaaba, babaaa, bababa\} \cup \{aaaaaaaa, \cdots\}$$

Regular languages :definition

We also note L_e for the language $\nu(e)$.

Definition

A language $L \subseteq \Sigma^*$ is called regular if and only if there exists some regular expression e over Σ such that

$$L = L_e.$$

Examples :
$L_1 = \{u \in \{a, b\}^* \mid |u| \text{ is even}\}$
This language is regular since : $L_1 = \{aa, ab, ba, bb\}^* = L_e$ for

$$e = \langle\langle\langle\langle\langle a \otimes a\rangle \oplus \langle a \otimes b\rangle\rangle \oplus \langle b \otimes a\rangle\rangle \oplus \langle b \otimes b\rangle\rangle \star \rangle$$

Regular languages :example

$L_2 = \{u \in \{a, b\}^* \mid u \text{ is square-free}\}$
$L_3 = \{u \in \{0, 1\}^* \mid$
u is the binary notation of an integer that is divisible by $4\}$
These languages are regular since :
$L_2 = \{\varepsilon, a, b, aa, ab, ba, bb, aba, bab\}$
$L_3 = \{0\} \cup \{1\} \cdot \{0, 1\}^* \cdot 00$

Regular languages :extended expressions

From now on, we accept as regular expressions, expressions using the usual symbols $\cup$ (instead of $\oplus$), $\cdot$ (instead of $\otimes$), using k-ary notation for the product and for the union (since these operations are associative). We add the symbol ε with value

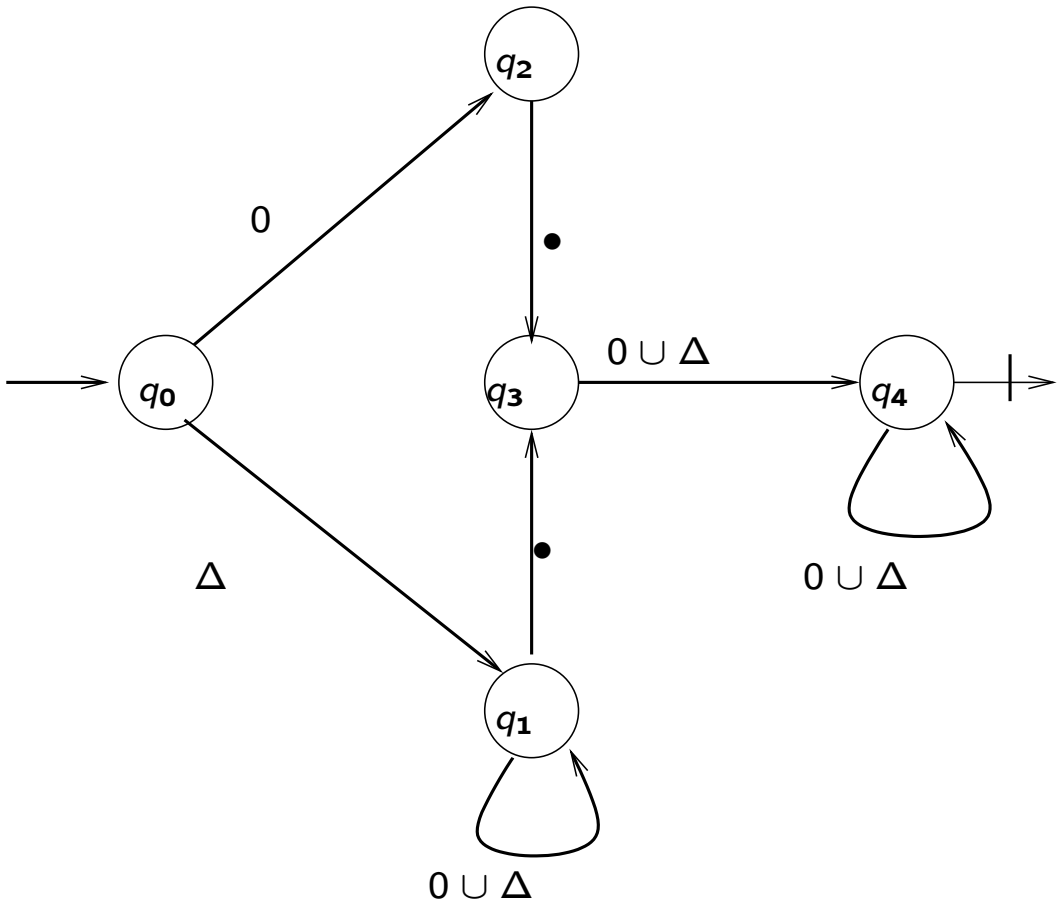
$$\nu(\varepsilon) = \nu(\emptyset^*) = \{\varepsilon\}.$$

For example :
$e = (a \cdot a \cdot a)^* \cdot (\varepsilon \cup b \cup (b \cdot b))$ or even more compactly
$f = (aaa)^* \cdot (\varepsilon \cup b \cup (bb))$.

Recognizable languages

Example

Let us describe the set of correct decimal integers.



where $\Delta = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

Definition

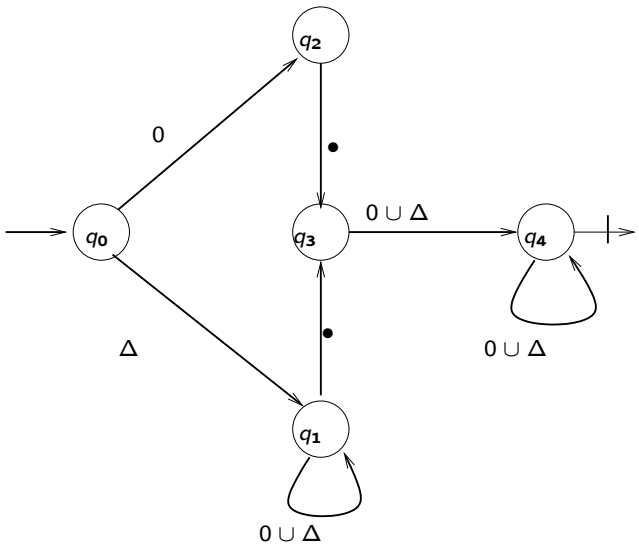
Definition

A deterministic finite automaton is a 5-tuple $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$ *where*

- Q *is a finite set, called the set of states*
- Σ *is an alphabet*
- $\delta : Q \times \Sigma \to Q$ *is a (partial) function called the transition function*
- $q_0 \in Q$ *is called the initial state*
- $F \subseteq Q$ *is the set of final states*

Σ is called the *input* alphabet. An automaton can be viewed as a device that, for every word $w \in \Sigma^*$, treats the word and eventually answers YES or NO.

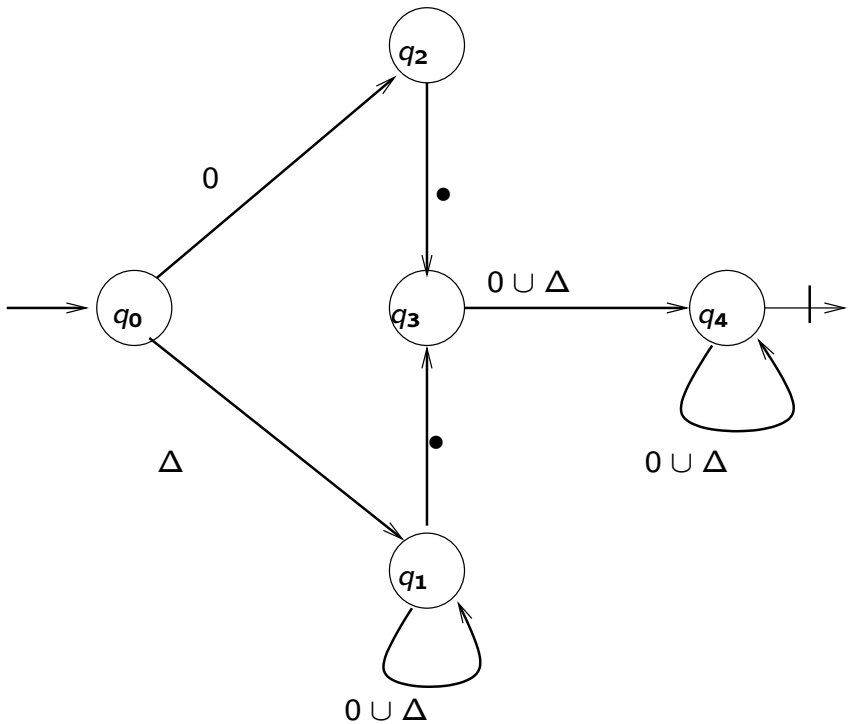
Example continued 2



Here $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$ with $Q = \{q_0, q_1, q_2, q_3, q_4\}$,
$\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \bullet\}$, $F = \{q_4\}$

Example continued 3

δ is described by the table :



$q \setminus x$	0	1	$\cdots$	9	•
q_0	q_2	q_1	q_1	q_1	—
q_1	q_1	q_1	q_1	q_1	q_3
q_2	—	—	—	—	q_3
q_3	q_4	q_4	q_4	q_4	—
q_4	q_4	q_4	q_4	q_4	—

dfa : Computations

We call computation of the dfa $\mathcal{A}$ every sequence :

$$u = (p_0, x_0, p_1)(p_1, x_1, p_2) \cdots (p_{\ell-1}, x_{\ell-1}, p_\ell)$$

where, $\forall i \in [0, \ell], p_i \in Q$, $\forall i \in [0, \ell - 1], x_i \in \Sigma$ and

$$\forall i \in [0, \ell - 1], \delta(p_i, x_i) = p_{i+1}.$$

The trace of the computation, $\mathrm{tr}(u)$ is the word :

$$w = x_0 x_1 \cdots x_{\ell-1}.$$

The computation u starts from p_0 and ends in state p_ℓ. We then note :

$$p_0 \xrightarrow{w}_{\mathcal{A}} p_\ell$$

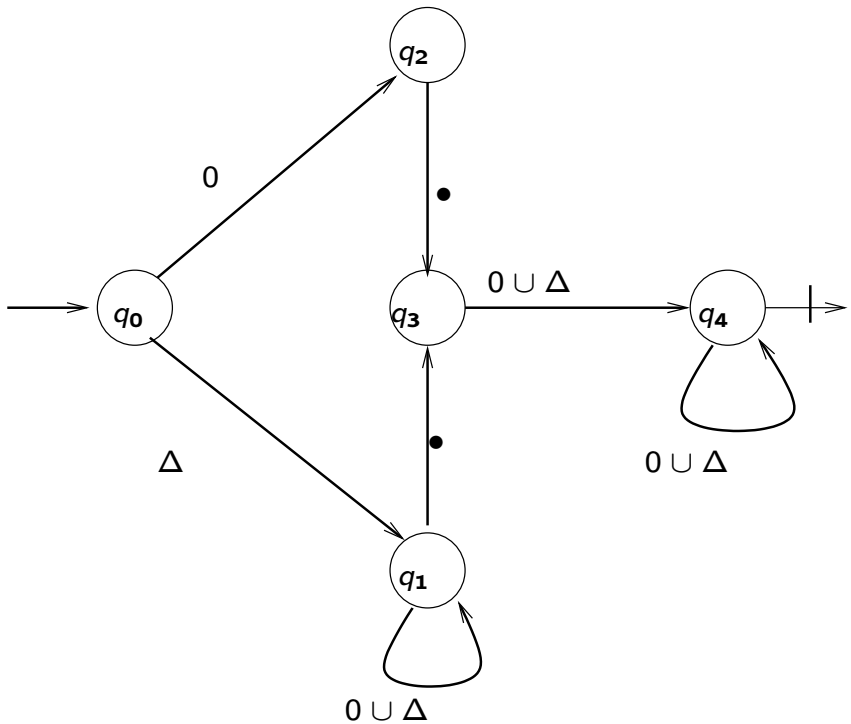
which can be read : "$\mathcal{A}$ moves from p_0 to p_ℓ reading w".

dfa : Computations

The language recognized by $\mathcal{A}$ is the set of all words $w \in \Sigma^*$ such that, there exists a computation of $\mathcal{A}$, starting in q_0, ending in some $q \in F$, with trace $\mathrm{tr}(u) = w$. More formally :

$$\mathrm{L}(\mathcal{A}) = \{w \in \Sigma^* \mid \exists q \in F, q_0 \xrightarrow{w}_{\mathcal{A}} q\}$$

dfa : Computations, examples

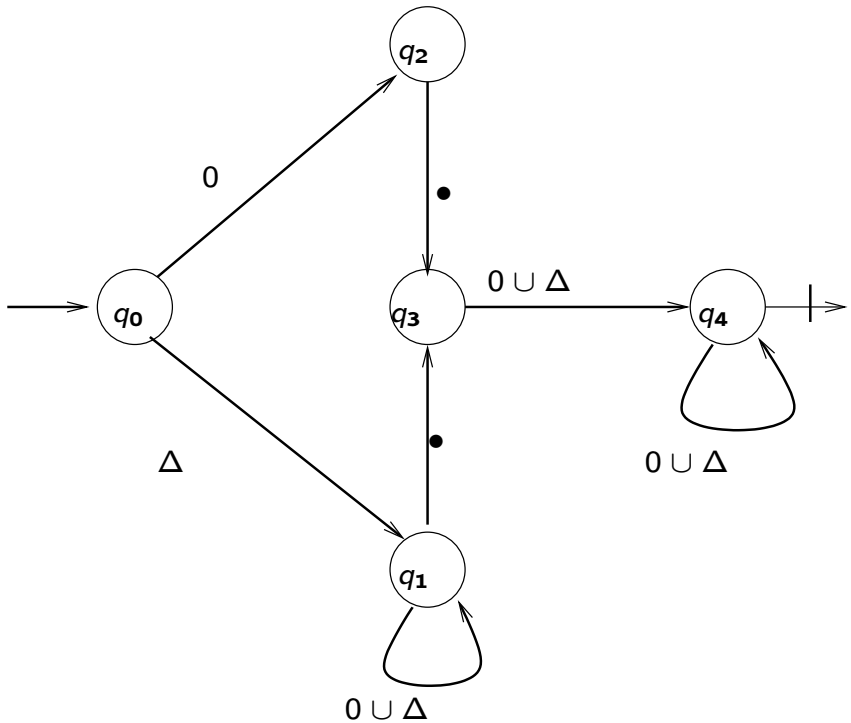


$$w_1 = 20 \bullet 25$$

computation : $(q_0, 2, q_1)(q_1, 0, q_1)(q_1, \bullet, q_3)(q_3, 2, q_4)(q_4, 5, q_4)$

Since q_4 is final, w_1 is accepted.

dfa : Computations, examples

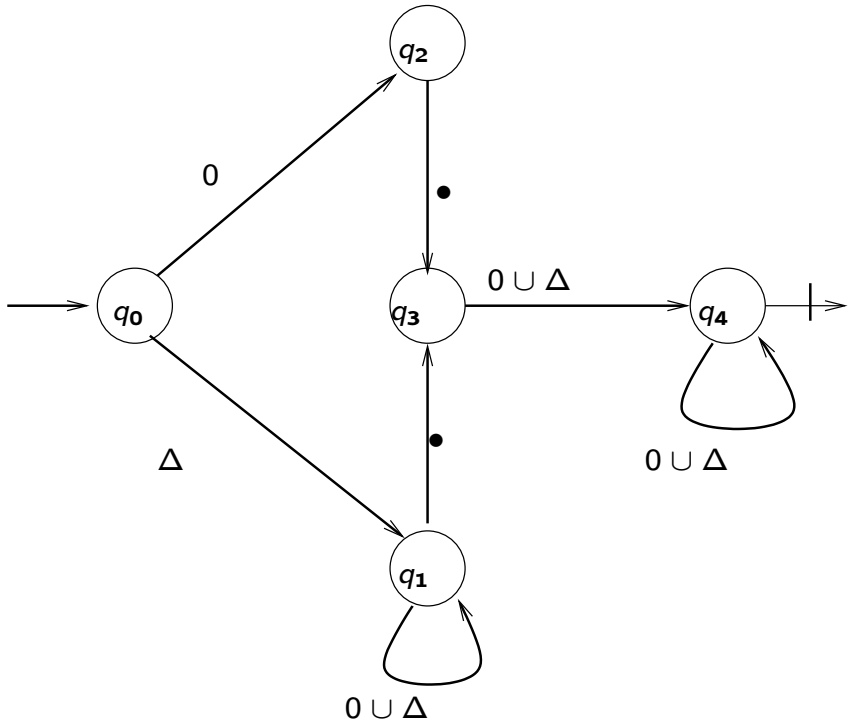


$$w_2 = 201$$

$$\text{computation} : (q_0, 2, q_1)(q_1, 0, q_1)(q_1, 1, q_1)$$

Since q_1 is not final, w_2 is rejected.

dfa : Computations, examples



$$w_3 = 21\bullet\bullet$$

computation : $(q_0, 2, q_1)(q_1, 0, q_1)(q_1, \bullet, q_3)$ and $\delta(q_3, \bullet)$ is undefined

Since no computation starting on q_0 can read w_3, the word w_3 is rejected.

Complete dfa

Let $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$ be a DFA. It is called complete if the transition function δ is a total map :

$$Q \times \Sigma \to Q.$$

In this case δ can be extended into a total map

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

by induction over the length of words : $\forall q \in Q, \forall x \in \Sigma, \forall w \in \Sigma^*$:

$$\delta^*(q, \varepsilon) = q$$

$$\delta^*(q, x) = \delta(q, x)$$

$$\delta^*(q, wx) = \delta(\delta^*(q, w), x)$$

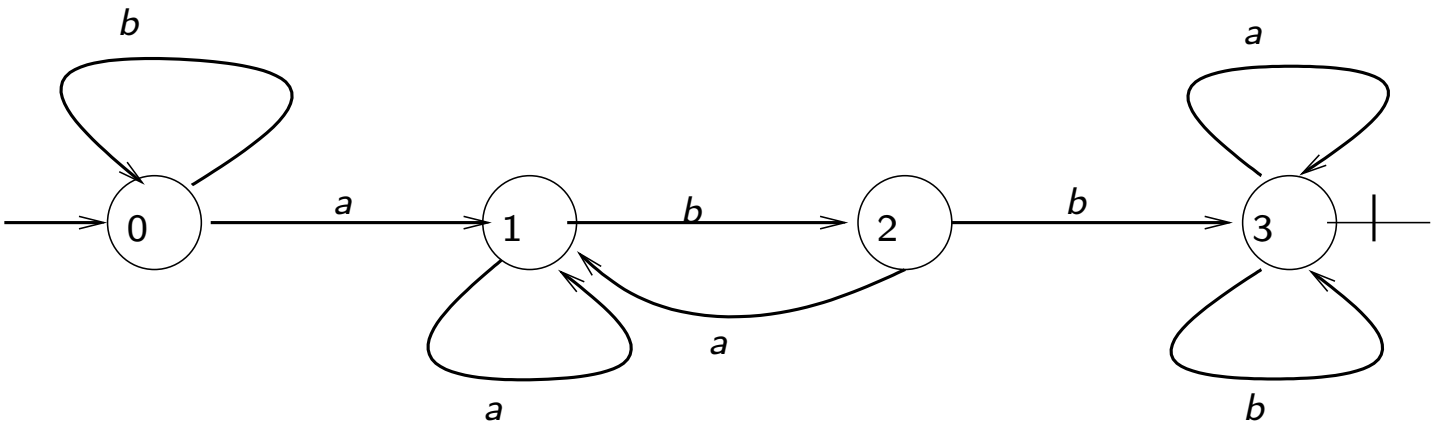
Complete dfa : programming δ^*

The following program (scheme) computes, for every input $w \in \Sigma^*$, the state $\delta^*(q_0, w)$ in linear time.

INPUT :$w = w[0]w[1]\cdots w[n-1]$.

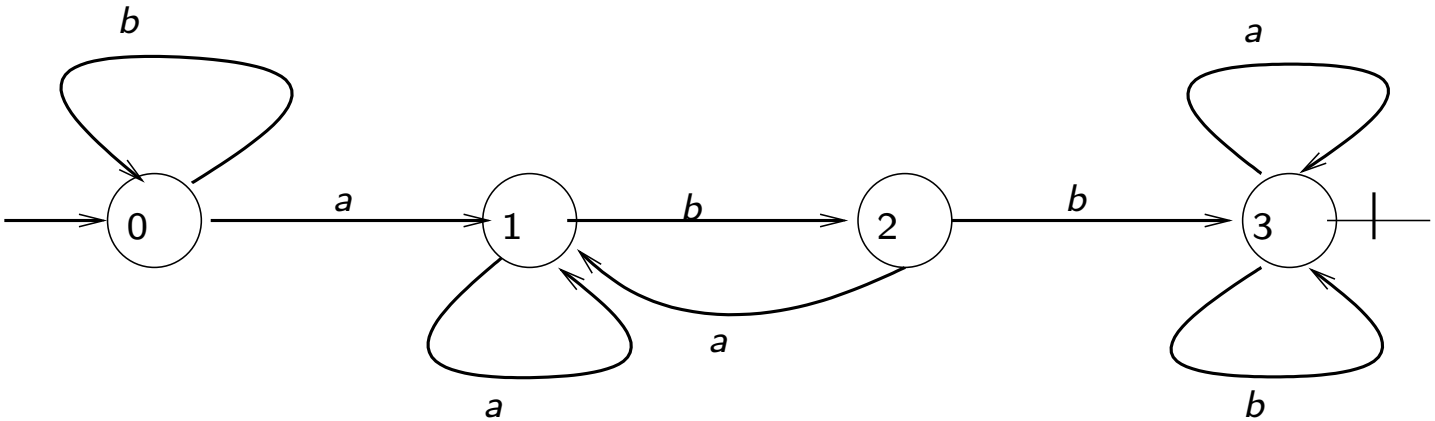
$q \leftarrow q_0$ { start with the initial state}
for $k \leftarrow 0$ to $n-1$ **do**
  $q \leftarrow \delta(q, w[k])$ { update the current state }
end for
return q

Complete dfa :example



This dfa is complete.
What is the language $\mathrm{L}(\mathcal{A})$?

Complete dfa :example

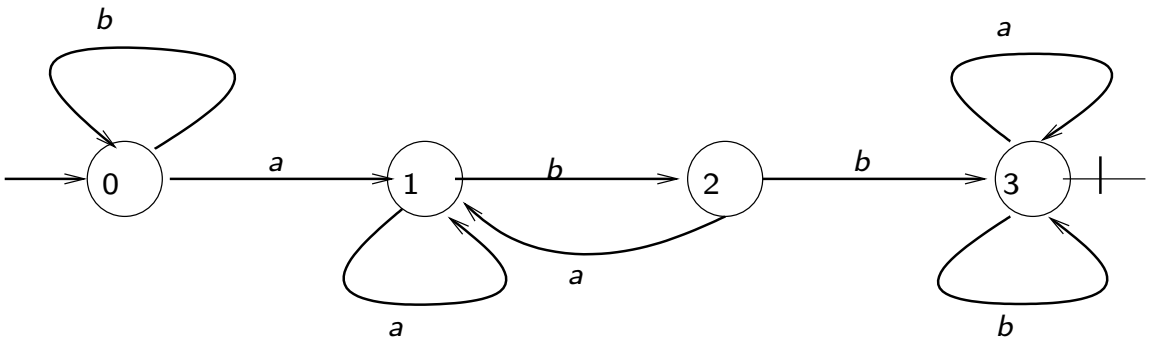


Computation over *aababbb* :

$$u_1 = (0, a, 1)(1, a, 1)(1, b, 2)(2, a, 1)(1, b, 2)(2, b, 3)(3, b, 3)$$

$\delta^*(0, aababbb) = 3$ and 3 is final $\rightarrow$ *aababbb* is recognized.

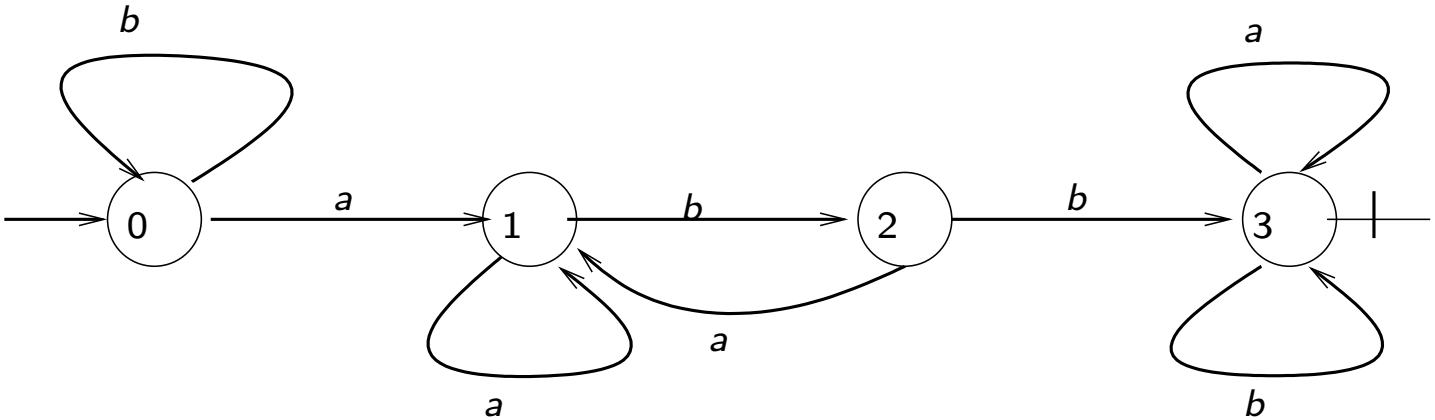
Complete dfa :example



u	$\delta^*(0,u)$
ε	0
b	0
a	1
a	1
b	2
a	1
b	2

$\delta^*(0, baabab) = 2$ and 2 is not final $\rightarrow$ $aababbb$ is not recognized.

Complete dfa :example



Let $\mathrm{LSP}(w)$ denote the longuest suffix of w which is prefix of abb.

$$\delta^*(0, w) = 3 \iff abb \text{ is factor of } w$$

$$\delta^*(0, w) = i \leq 2 \Leftrightarrow |\mathrm{LSP}(w)| = i.$$

This can be proved by induction on the length of w.
Hence $\mathrm{L}(\mathcal{A}) = (a \cup b)^* \cdot abb \cdot (a \cup b)^*$.

dfa completion

Proposition

For every dfa $\mathcal{A}$, *one can construct, in linear time, a complete* dfa $\mathcal{A}'$ *such that* $\mathrm{L}(\mathcal{A}) = \mathrm{L}(\mathcal{A}')$

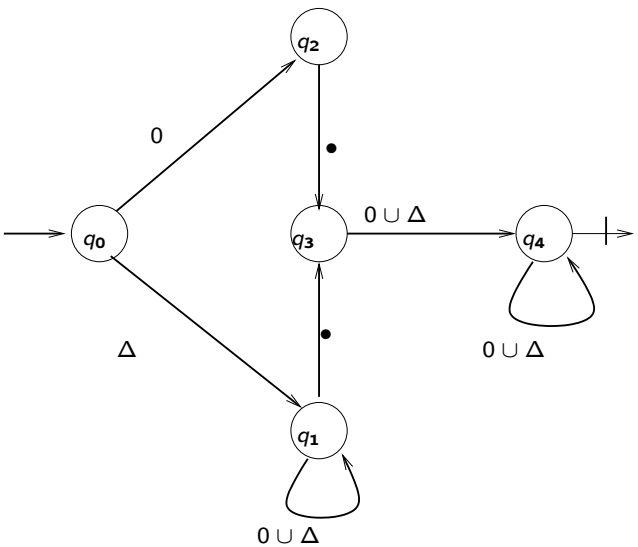
Proof : Let $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$ be a non-complete dfa.
We build a new dfa $\mathcal{A}'$ from $\mathcal{A}$, by adding a "sink" P to the set of states.

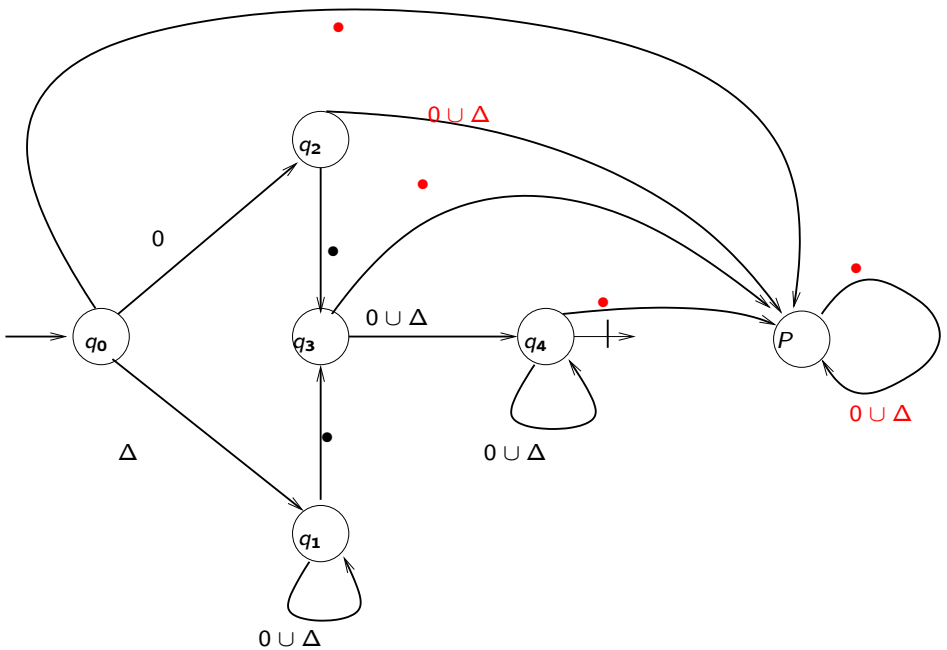
$$\mathcal{A}' := \langle Q', \Sigma, \delta', q_0, F \rangle$$

$Q' := Q \cup \{P\}$ where $P \notin Q$ and $\delta' : Q' \times \Sigma \to Q'$ is defined by :

$$\begin{aligned} \delta'(q,x) &= \delta(q,x) \text{ if } q \in Q, x \in \Sigma, \text{ and } (q,x) \in \mathrm{dom}(\delta) \\ \delta'(q,x) &= P \text{ if } q \in Q, x \in \Sigma \text{ and } (q,x) \notin \mathrm{dom}(\delta) \\ \delta'(P,x) &= P \text{ if } x \in \Sigma. \end{aligned}$$

Completing a dfa : example





The completed automaton.

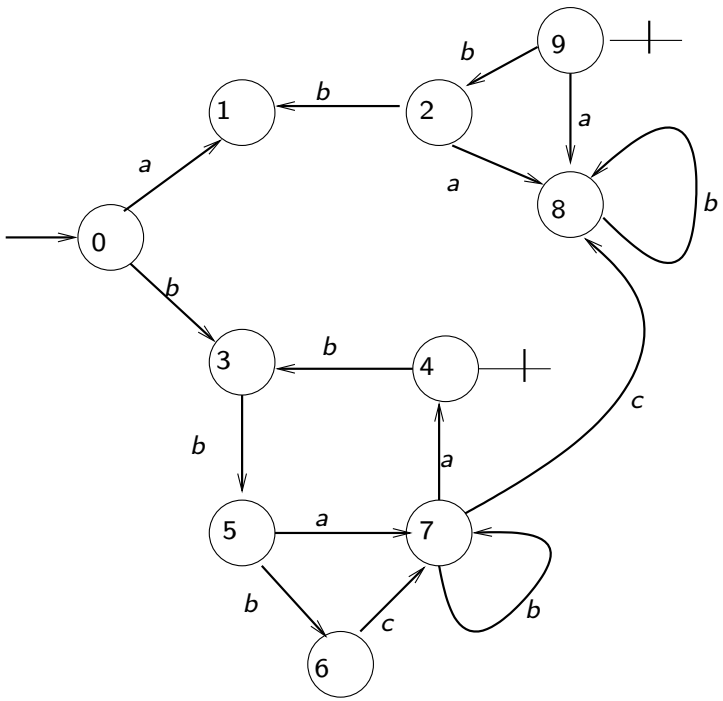
Reachability in dfa

A state q is called reachable from state p if there exists some word w such that

$$p \xrightarrow{w}_{\mathcal{A}} q$$

A state q is called accessible if it is reachable from q_0
A state q is called co-accessible if some final state $p \in F$ is reachable from q
A state q is called useful if it is both accessible and co-accessible

Reachability :example



$0 \xrightarrow{bbac}_{\mathcal{A}} 8 : 8$ is reachable from 0 ; $9 \xrightarrow{bb}_{\mathcal{A}} 1 : 1$ is reachable from 9

$6 \xrightarrow{ca}_{\mathcal{A}} 4 : 4$ is reachable from 6 ; $0 \xrightarrow{bb}_{\mathcal{A}} 6 : 6$ is reachable from 0

Hence : $0, 8, 6$ are accessible, $6, 9$ are co-accessible.

Trim dfa

Definition

A Deterministic Finite Automaton $\mathcal{A}$ is called trim if every state of $\mathcal{A}$ is useful.

Proposition (trim normal form)

For every dfa $\mathcal{A}$ one can achieve in linear time the following
1- test whether $\mathrm{L}(\mathcal{A}) \neq \emptyset$
2- if $\mathrm{L}(\mathcal{A}) \neq \emptyset$, *then construct a trim* dfa $\mathcal{A}'$ *such that* $\mathrm{L}(\mathcal{A}) = \mathrm{L}(\mathcal{A}')$

Making trim a dfa

Proof of the proposition :
Let $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$.
Let Q_1 (respectively Q_2) be the set of accessible (resp. co-accessible) states .
0- We compute $\hat{Q} = Q_1 \cap Q_2$ (the set of useful states).
1- $q_0 \in \hat{Q}$ if and only if $\mathrm{L}(\mathcal{A}) \neq \emptyset$
2- In the case where $q_0 \in \hat{Q}$, we let

$$\hat{\mathcal{A}} = \langle \hat{Q}, \Sigma, \hat{\delta}, q_0, \hat{F} \rangle$$

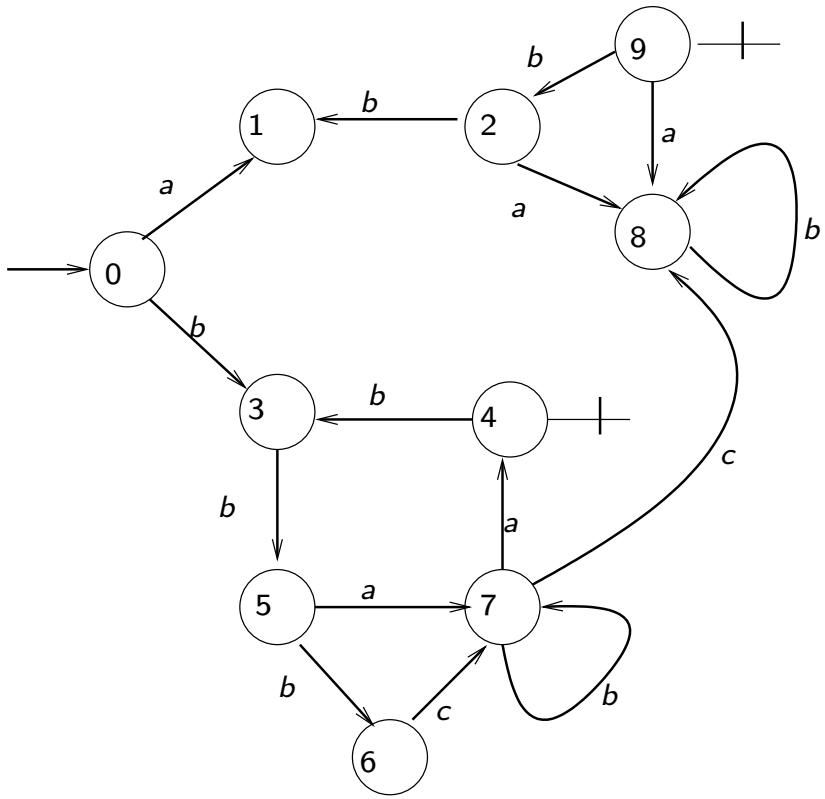
where $\hat{\delta} = \delta \restriction \hat{Q} \times \Sigma$ (the restriction of δ on useful states), $\hat{F} = F \cap \hat{Q}$.

Making trim a dfa

point 0 :
Q_1 (resp. Q_2) can be computed by a depth-first search, from q_0 (resp. F), in the oriented graph $\langle Q, E\rangle$ (resp. $\langle Q, E^{-1}\rangle$) where

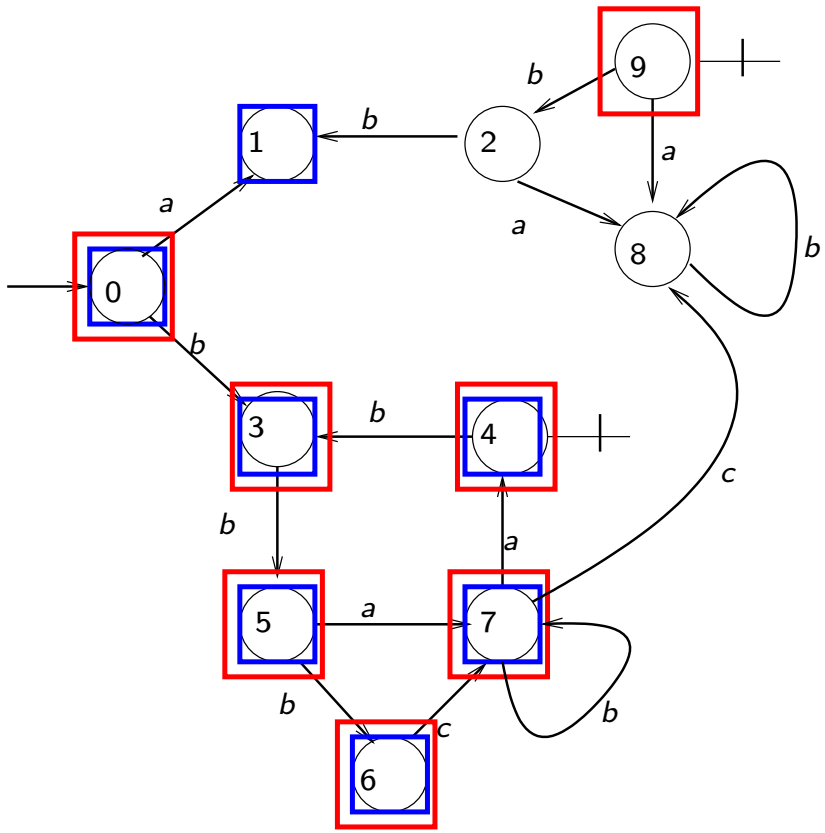
$$E = \{(q, q') \mid \exists x \in \Sigma, \delta(q, x) = q'\}$$

Making trim a dfa :example



Let $\mathcal{A}$ be the above dfa .

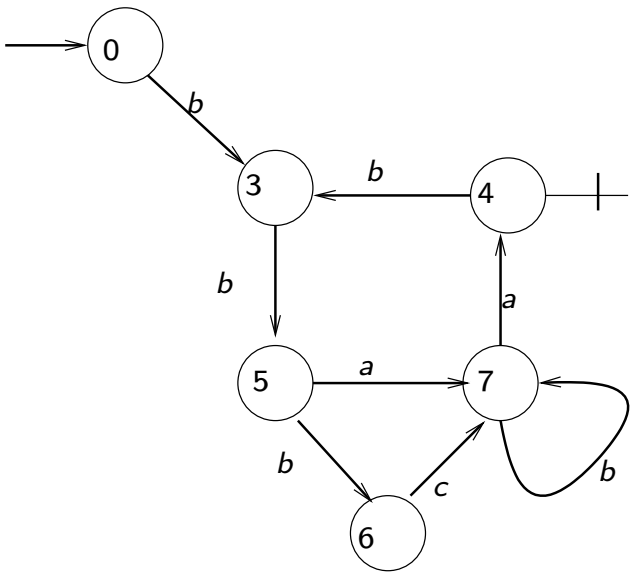
Making trim a dfa :example



In this example :

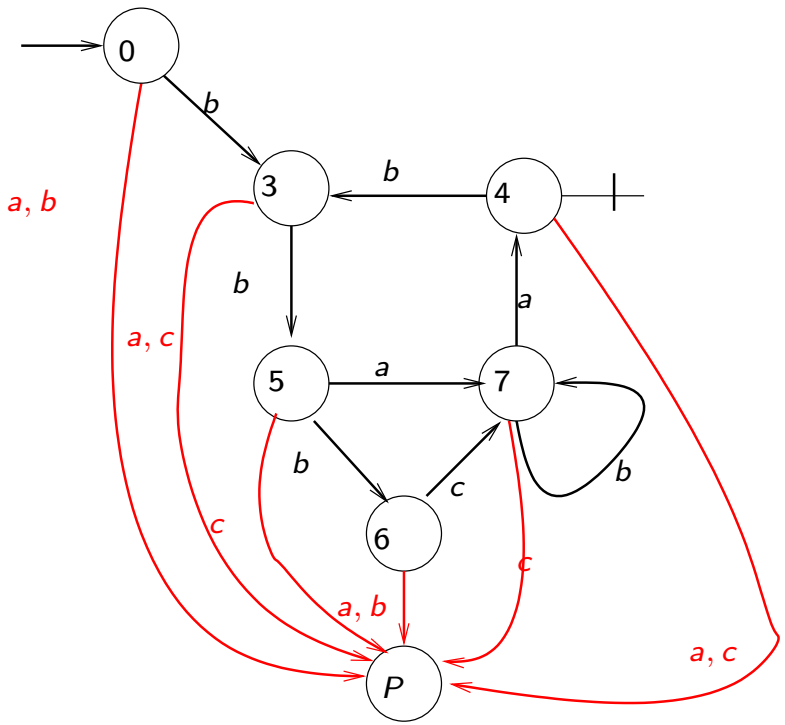
$$Q_1 = \{0, 1, 3, 5, 6, 7, 4\}, Q_2 = \{4, 7, 5, 6, 3, 0, 9\}$$

Making trim a dfa :example



We obtain the trim automaton $\hat{\mathcal{A}}$.

completion of the trim dfa



$\hat{\mathcal{A}}'$: the completion of the trim automaton $\hat{\mathcal{A}}$.

Minimal dfa

Definition

Let $\mathcal{A}$ be some complete dfa *. We call it minimal if, for every complete* dfa $\mathcal{B}$*, if* $\mathrm{L}(\mathcal{A}) = \mathrm{L}(\mathcal{B})$ *then $\mathcal{A}$ has fewer states than $\mathcal{B}$.*

Theorem

Let $L \subseteq \Sigma^*$ be some recognizable language.
1- There exists a minimal complete dfa *recognizing L*
2- If two complete dfa $\mathcal{A}, \mathcal{B}$ *are minimal and recognize L, then these two automata are isomorphic (i.e. $\mathcal{B}$ can be obtained from $\mathcal{A}$ just by state-renaming).*

NB1 : point 1 is obvious : just take, among the complete dfa recognizing L, one which has the smallest number of states.
NB2 : point 2 is not obvious ; we shall see later the main arguments that prove this statement.

Minimization of a dfa : method

Let $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$ be a complete dfa. Let us sketch a method for computing the unique minimal dfa $\mathcal{M}$ which is equivalent with $\mathcal{A}$.

step 1 :We compute an equivalence relation $\equiv$ over Q (the "Nerode equivalence")

step 2 : We build the quotient automaton $\mathcal{M} = \mathcal{A}/\equiv$ by merging all the states that belong to the same equivalence class.

Nerode equivalence

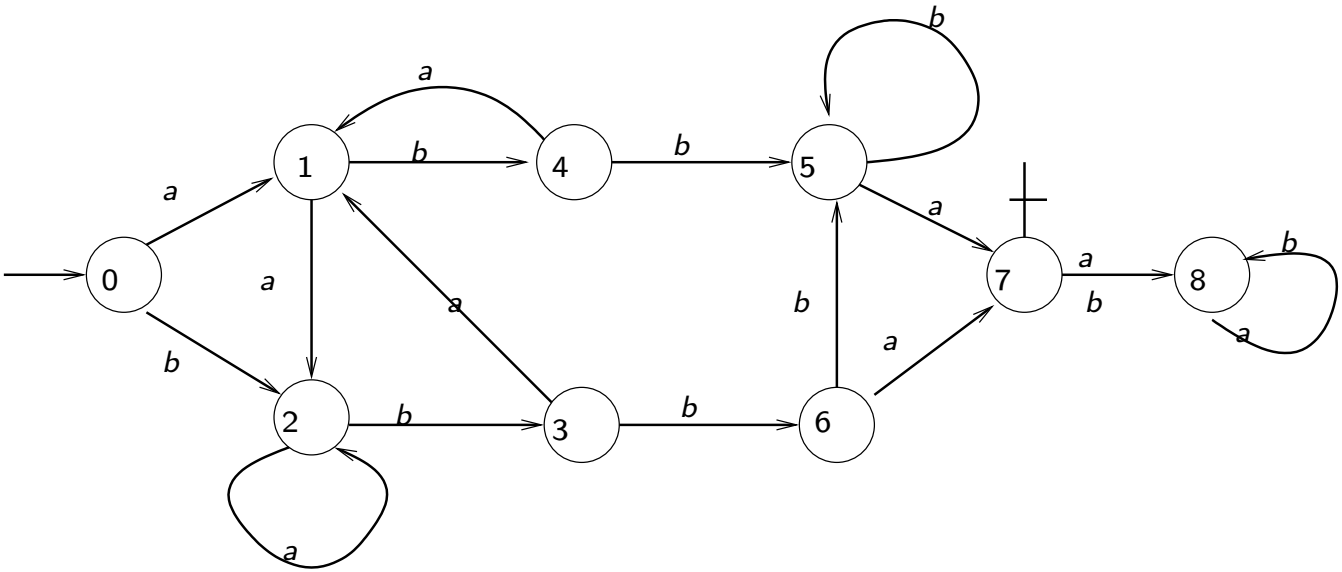
We define an equivalence relation $\equiv$ over Q as follows.

Definition

For every states $p, q \in Q$, $p \equiv q$ iff

$$\forall u \in \Sigma^*, \delta^*(p, u) \in F \Leftrightarrow \delta^*(q, u) \in F.$$

Nerode equivalence :example



One easily checks that $5 \equiv 6$.
$\delta^*(1, bba) = 7 \in F$ while $\delta^*(0, bba) = 1 \notin F$ hence $0 \not\equiv 1$.

Nerode equivalence : computation

We compute a decreasing sequence of equivalences $\equiv_i$ over Q ;

$$\equiv_0 := \{(q, q') \mid (q \in F \text{ and } q' \in F) \text{ or } (q \notin F \text{ and } q' \notin F)\}$$

$$\equiv_{i+1} := \{(q, q') \mid \forall x \in X \cup \{\epsilon\}, \delta^*(q, x) \equiv_i \delta^*(q', x)\}$$

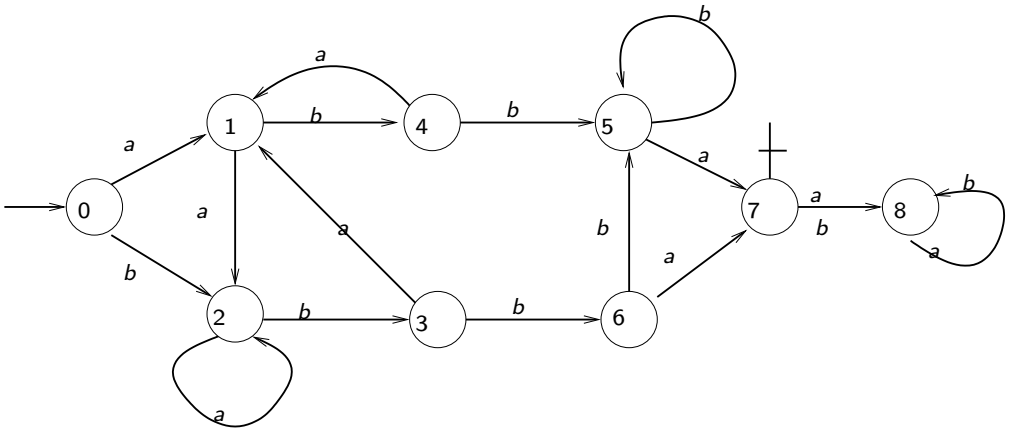
For some $n \leq |Q|$:

$$\equiv_n = \equiv_{n+1} .$$

The Nerode equivalence is :

$$\equiv = (\bigcap_{k=0}^{\infty} \equiv_k) = \equiv_n$$

Minimization of a dfa : example



Nerode equivalence :

$\equiv_0 := \{\{0,1,2,3,4,5,6,8\},\{7\}\},$

$\equiv_1 := \{\{0,1,2,3,4,8\},\{5,6\},\{7\}\}$

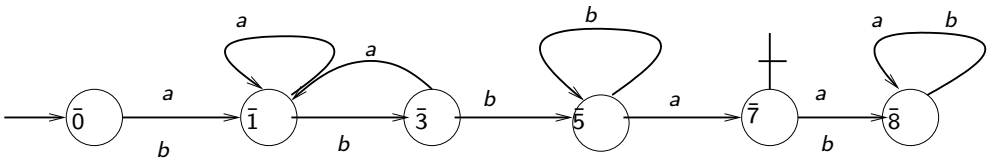
$\equiv_2 := \{\{0,1,2,8\},\{3,4\},\{5,6\},\{7\}\},$

$\equiv_3 := \{\{0,8\},\{1,2\},\{3,4\},\{5,6\},\{7\}\},$

$\equiv_4 := \{\{0\},\{8\},\{1,2\},\{3,4\},\{5,6\},\{7\}\}$

$\equiv_5 = \equiv_4$, hence $\equiv = \equiv_4$

Minimization of a dfa : example



Quotient automaton :obtained by merging

$$\bar{0} = \{0\}, \bar{1} = \{1, 2\}, \bar{3} = \{3, 4\}, \bar{5} = \{5, 6\}, \bar{7} = \{7\}, \bar{8} = \{8\}$$

$$\bar{\delta}(\bar{q}, x) := \overline{\delta(q, x)}$$

Minimization of a dfa : final algorithm

A $\rightarrow$ B (trim) $\rightarrow$ C (complete) $\rightarrow$ D (minimal complete).