Formal Languages-Course 3.

Géraud Sénizergues

Bordeaux university

11/05/2020

Master computer-science MINF19, IEI, 2019/20

contents

Minimal complete deterministic automaton

Minimal dfa :proofs

We asserted

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).

Let us now prove point 2 of the theorem. The proof will also explain why the minimization algorithm is correct.

Minimal dfa : a candidate

We denote by $\mathbb{Q}(L)$ the set of left-quotients of L :

$$\mathbb{Q}(L) := \{u^{-1}L \mid u \in \Sigma^*\}.$$

Let $L \subseteq \Sigma^*$. We define the deterministic automaton :

$$\mathcal{M} = \langle \Sigma, \mathbb{Q}(L), L, R, \theta \rangle$$

where
- L is the initial state
- $R = \{P \in \mathbb{Q}(L) \mid \epsilon \in P\}$ is the set of final states
- $\theta : (P, x) \mapsto x^{-1}P$ is the transition function

By induction over $|w|$: $\forall w \in \Sigma^*, \theta^*(L, w) = w^{-1}L$. Hence

$$\mathrm{L}(\mathcal{M}) = L.$$

Minimal dfa : 3 properties

We shall show that
M1- $\mathcal{M}$ is a finite deterministic automaton recognizing L,
M2- $\mathcal{M}$ is minimal
M3- every accessible complete dfa $\mathcal{A}$ recognizing L is such that $\mathcal{A}/\equiv \approx \mathcal{M}$ where $\equiv$ is the Nerode equivalence over $\mathcal{A}$.

Minimal dfa : an automaton homomorphism

Let $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$ be a complete dfa recognizing L. We assume every state of $\mathcal{A}$ is accessible.
For every state $q \in Q$, we note

$$\mathrm{L}(q, \mathcal{A}) := \{w \in \Sigma^* \mid \delta^*(q, w) \in F\}.$$

Let $\varphi : Q \to \mathcal{P}(\Sigma^*)$ defined by :

$$\varphi(q) = \mathrm{L}(q, \mathcal{A}).$$

Minimal dfa : an automaton homomorphism

Fact

$\forall q \in Q, \forall w \in \Sigma^*, \varphi(\delta(q,w)) = w^{-1}\varphi(q).$

Proof:

$$\begin{aligned}
\varphi(\delta(q,x)) &= \mathrm{L}(\delta(q,x),\mathcal{A}) \\
&= \{w \in \Sigma^* \mid \delta^*(\delta(q,x),w) \in F\} \\
&= \{w \in \Sigma^* \mid \delta^*(q,xw) \in F\} \\
&= \{w \in \Sigma^* \mid xw \in \mathrm{L}(q,\mathcal{A})\} \\
&= x^{-1}\mathrm{L}(q,\mathcal{A}) \\
&= x^{-1}\varphi(q).
\end{aligned}$$

By induction over $|w|$, we get the fact. □

Minimal dfa : an automaton homomorphism

If $\delta^*(q_0, w) = q$ then

$$\varphi(q) = \varphi(\delta^*(q_0, w)) = w^{-1}\mathrm{L}(q_0, \mathcal{A}).$$

$$\varphi(q) = w^{-1}L \in \mathbb{Q}(L).$$

Since $\mathcal{A}$ has only accessible states, $\varphi : Q \to \mathbb{Q}(L)$.Moreover, every right-quotient $w^{-1}\mathrm{L}(q_0, \mathcal{A})$ belongs to the image of φ, hence

$$\varphi : Q \twoheadrightarrow \mathbb{Q}(L) \text{ is surjective },$$

showing that

$$\mathrm{Card}(Q) \geq \mathrm{Card}(\mathbb{Q}(L)).$$

Hence $\mathbb{Q}(L)$ is finite (M1) and $\mathcal{M}$ is minimal (M2).

Minimal dfa : an automaton homomorphism

The map $\varphi : Q \to \mathbb{Q}(L)$ has the three properties :

$$\varphi(\delta(q,x)) = \theta(\varphi(q),x) \quad (1)$$

$$\varphi(q_0) = L \quad (2)$$

$$q \in F \Leftrightarrow \varphi(q) \in R \quad (3)$$

Property (1) is a reformulation of the above fact 1. Properties (2,3) are easily checked.
The map $\varphi : Q \to \mathbb{Q}(L)$ is called an *automaton-homomorphism* from $\mathcal{A}$ into $\mathcal{M}$.

Minimal dfa : an automaton homomorphism

Note that for every $q, q' \in Q$,

$$q \equiv q' \Leftrightarrow \varphi(q) = \varphi(q').$$

Let us consider the quotient automaton $\mathcal{A}/\equiv$ obtained by merging the equivalent states :

$$\mathcal{A}/\equiv = \langle \Sigma, \bar{Q}, \bar{q_0}, \bar{F}, \bar{\delta} \rangle$$

where :
- $\bar{Q} = \{[q]_\equiv \mid q \in Q\}$
- $\bar{q_0} = [q_0]_\equiv$
- $\bar{F} = \{[q]_\equiv \mid q \in F\}$
- $\bar{\delta}([q]_\equiv, x) = \overline{\delta(q, x)}$

The map $\bar{\varphi} : \bar{Q} \to \mathbb{Q}(L)$ defined by : $\bar{\varphi}([q]_\equiv) = \varphi(q)$ is well-defined, is bijective and is also an automaton-homomorphism. Hence it is an automaton-isomorphism i.e. (M3).

Non-deterministic finite automata

Non-deterministic finite automata : motivation

We define a more general notion of finite automaton. But the class of recognized languages is still the same.
This makes easier the task of proving that a language is recognizable.
1- For a given language one can build a smaller automaton
2- For a given language one can find easily a non-deterministic automaton, while finding directly a deterministic automaton might be difficult

Non-deterministic finite automata : motivation

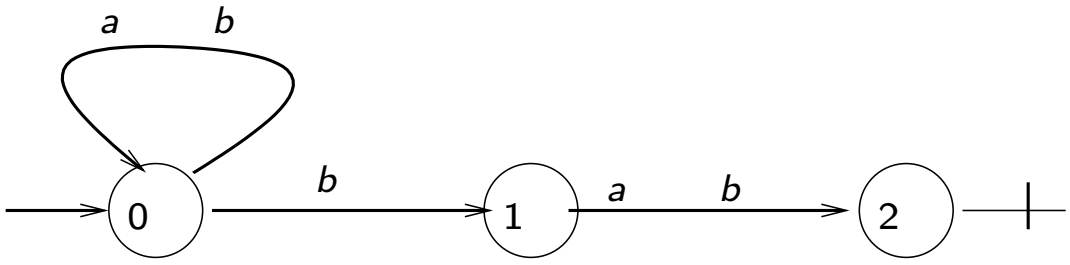
Example

$L_1 = \{a, b\}^* \cdot b \cdot \{a, b\}, \quad L_2 = a(ba)^* \cup (abb)^* a$
$L_3 = \{a, b\}^* \setminus [a(ba)^* \cup (abb)^* a]$

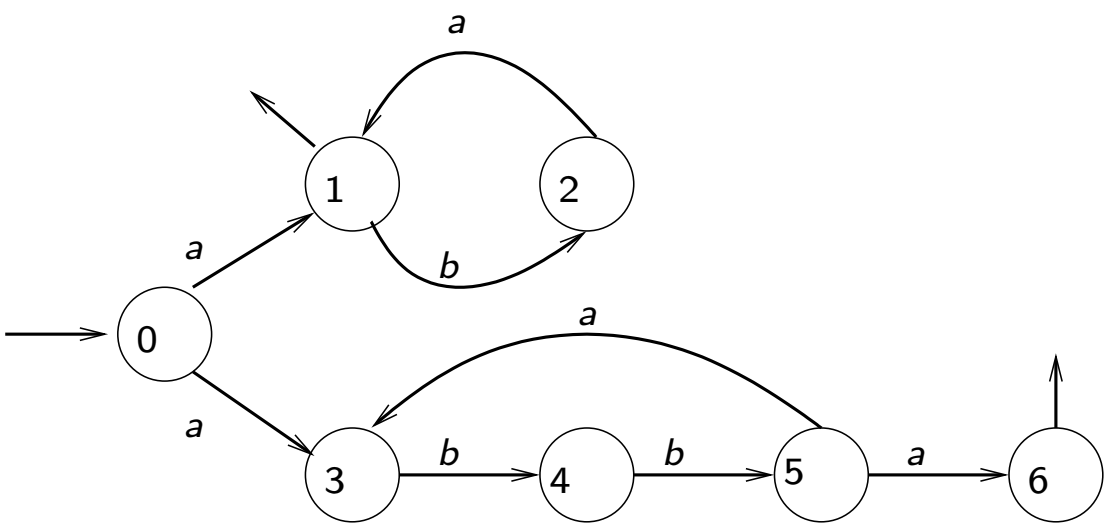
For L_1, L_2 : it is clear that they are regular. Are they recognizable ?
For L_3 : is it regular ? is it recognizable ?

Non-deterministic finite automata : examples

Some non-deterministic automata for L_1, L_2 :



$\mathcal{A}_1$



$\mathcal{A}_2$

Non-deterministic finite automata : definition

Definition

A non-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*
- $\delta \subseteq Q \times \Sigma \times Q$ *is the set of transitions*
- $q_0 \in Q$ *is called the initial state*
- $F \subseteq Q$ *is the set of final states*

NB : a non-deterministic automaton might be ... deterministic !
What "non-deterministic" actually means is : not promised to be deterministic.

nfa : Computations

We call computation of the nfa $\mathcal{A}$ 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

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

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".

nfa : 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 :

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

determinization : example

Let us consider the nfa

$$\mathcal{A}_1 = \langle Q_1, \Sigma, \delta_1, q_0, F \rangle$$

with $Q_1 = \{0, 1, 2\}$, $\Sigma = \{a, b\}$,
$\delta_1 = \{(0, a, 0), (0, b, 0), (0, b, 1), (1, a, 2), (1, b, 2)\}$, $F = \{2\}$.

determinization : example

The determinized automaton :

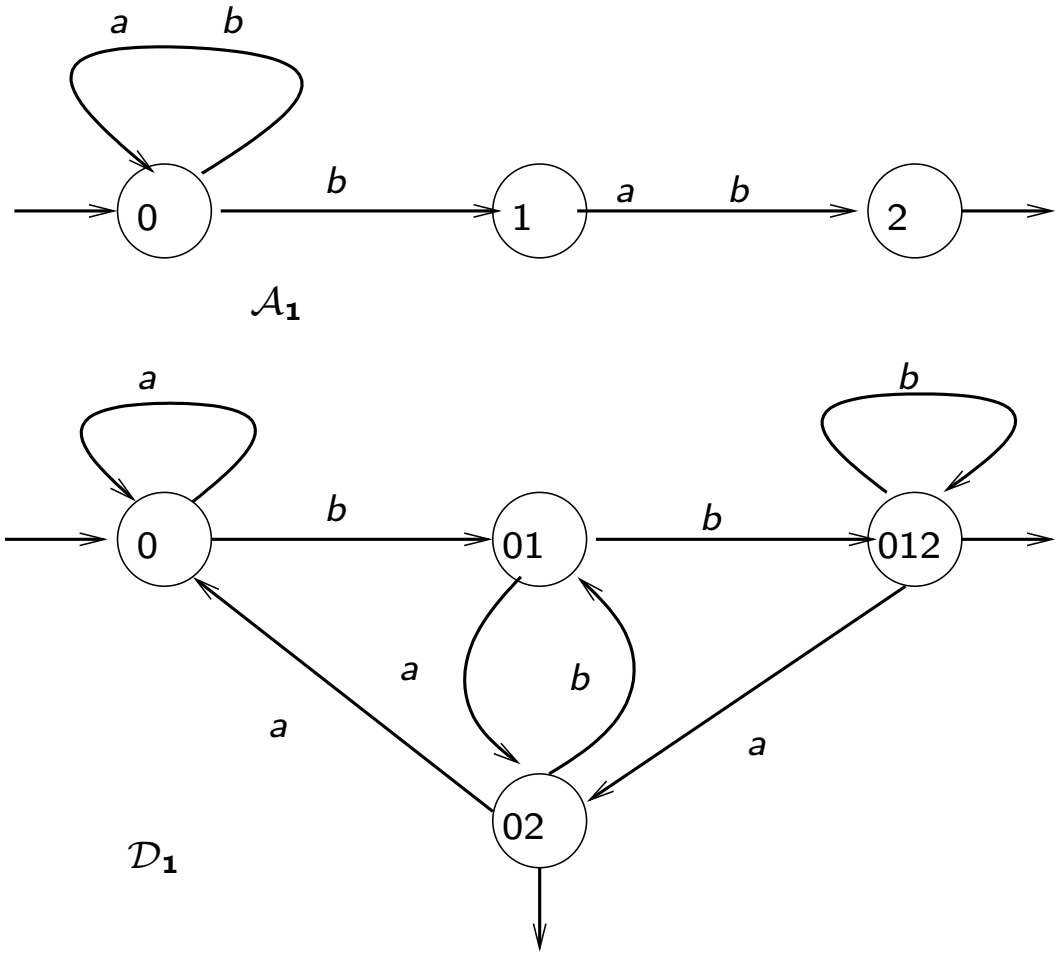
$$\mathcal{D}_1 = \langle Q_1', \Sigma, \delta_1', q_0', F_1' \rangle$$

with $Q_1' = \{\{0\}, \{0,1\}, \{0,2\}, \{0,1,2\}\}$, $\Sigma = \{a, b\}$, $F_1' = \{\{0,2\}, \{0,1,2\}\}$. The map δ_1' is described by :

$q \setminus x$	a	b
0	0	1
01	02	012
02	0	01
012	02	012

determinization : example

The determinized automaton $\mathcal{D}_1$:



determinization : the theorem

Theorem

For every non-deterministic finite automaton $\mathcal{A}$ there exists a deterministic finite automaton $\mathcal{D}$, which can be constructed from $\mathcal{A}$, such that

$$\mathrm{L}(\mathcal{A}) = \mathrm{L}(\mathcal{D}).$$

Sketch of proof

Let $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$. We build $\mathcal{D} = \langle \mathcal{P}(Q), \Sigma, \Delta, \{q_0\}, \mathcal{F} \rangle$ where, for every $P \subseteq Q, x \in \Sigma$

$\Delta(P, x) = \{q \in Q \mid \exists p \in P, (p, x, q) \in \delta\}$

$\mathcal{F} = \{P \in \mathcal{P}(Q) \mid P \cap F \neq \emptyset\}$.

We can prove by induction over $|u|$ that :

$$\Delta^*(P, u) = \{q \in Q \mid \exists p \in P, p \xrightarrow{u}_{\mathcal{A}} q\}.$$

determinization : the theorem

$$\Delta^*(P, u) = \{q \in Q \mid \exists p \in P, p \xrightarrow{u}_{\mathcal{A}} q\}.$$

It follows that :

$$\begin{aligned} u \in \mathrm{L}(\mathcal{A}) &\Leftrightarrow \exists q \in F, q_0 \xrightarrow{u}_{\mathcal{A}} q \\ &\Leftrightarrow \Delta^*(P, u) \cap F \neq \emptyset \\ &\Leftrightarrow \Delta^*(P, u) \in \mathcal{F}. \end{aligned}$$

□

determinization : the theorem

Exercice

Build $\mathrm{dfa}\ \mathcal{D}_2, \mathcal{D}_3$ *recognizing the languages*

$$L_2 = a(ba)^* \cup (abb)^* a, \quad L_3 = \{a, b\}^* \setminus [a(ba)^* \cup (abb)^* a]$$

Kleene's theorem

Kleene's theorem

For every finite alphabet we denote by :
- $\mathrm{REG}(\Sigma^*)$ the set of regular languages over Σ
- $\mathrm{REC}(\Sigma^*)$ the set of recognizable languages over Σ

Theorem

For every finite alphabet Σ, $\mathrm{REG}(\Sigma^*) = \mathrm{REC}(\Sigma^*)$.

The proof is constructive :
- we give an algorithm translating every $\mathrm{nfa}\ \mathcal{A}$ into a regular expression e, such that $\mathrm{L}(\mathcal{A}) = L_e$.
- we give an algorithm translating every regular expression e into a $\mathrm{nfa}\ \mathcal{A}$ such that $\mathrm{L}(\mathcal{A}) = L_e$.

BMC-algorithm

The BMC-algorithm (Brzozowski-Mc-Kluskey) : translates every nfa $\mathcal{A}$ into a regular expression e ;

Three steps :

step 1 : normalisation of $\mathcal{A}$: adds two new states.

step 2 : sequence of extended finite automaton, with strictly decreasing number of states.

step 3 : last extended automaton has only one transition : its label is e.

BMC-algorithm :e-automata

An extended-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
- $\delta \subseteq Q \times \mathrm{RE}(\Sigma) \times Q$ is the set of transitions
- $q_0 \in Q$ is called the initial state
- $F \subseteq Q$ is the set of final states

BMC-algorithm :e-automata

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

$$u = (p_0, e_0, p_1)(p_1, e_1, p_2)\cdots(p_{\ell-1}, e_{\ell-1}, p_\ell)$$

where, $\forall i \in [0, \ell], p_i \in Q$, $\forall i \in [0, \ell-1], e_i \in \mathrm{RE}(\Sigma)$ and

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

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

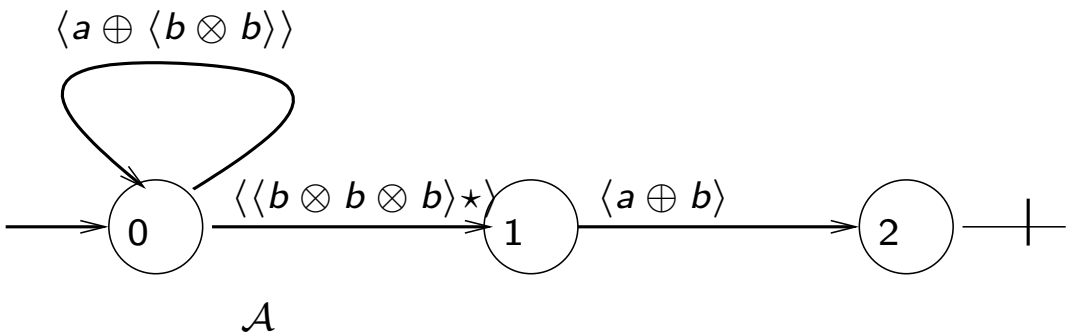
$$e = e_0 e_1 \cdots e_{\ell-1}.$$

The language recognized by $\mathcal{A}$ is defined by :

$$\begin{aligned} \mathrm{TR}(\mathcal{A}) &= \{e \in \mathrm{RE}(\Sigma) \mid \exists q \in F, q_0 \xrightarrow{e}_{\mathcal{A}} q\} \\ \mathrm{L}(\mathcal{A}) &= \bigcup_{e \in \mathrm{TR}(\mathcal{A})} \nu(e). \end{aligned}$$

BMC-algorithm :e-automata

Example :

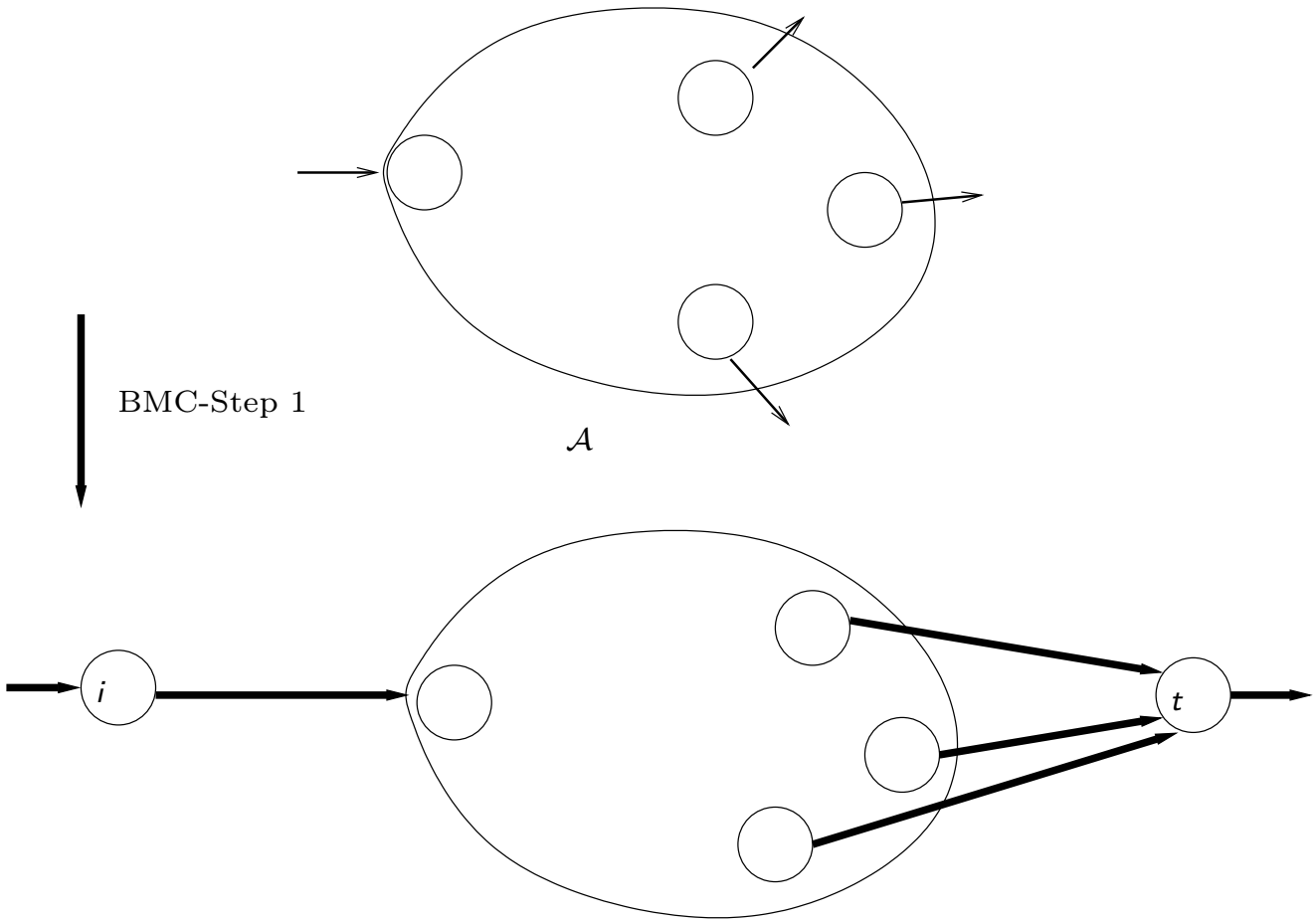


$$\mathrm{TR}(\mathcal{A}) = (\langle a \oplus \langle b \otimes b \rangle \rangle)^* \cdot \langle \langle b \otimes b \otimes b \rangle \star \rangle \cdot \langle a \oplus b \rangle.$$

$$\mathrm{L}(\mathcal{A}) = (a \cup (bb))^* \cdot (bbb)^* \cdot (a \cup b).$$

BMC-algorithm : step 1

Step 1 :



BMC-algorithm : step 2

The current efa is $\mathcal{C} = \langle Q_C, \Sigma, \delta_C, i, \{t\}\rangle$. We assume there is at most one transition from any state p to any other state p' (just make the union of all the labels (p, e, p') if there are several such transitions).
We remove a state $q \in Q_C \setminus \{i, t\}$.

BMC-algorithm : step 2

let $p_1, \cdots p_j \cdots p_h$ be the states in $Q_C \setminus \{q\}$ with transitions

$$(p_j, e_j, q)$$

let $r_1, \cdots r_k \cdots r_\ell$ be the states in $Q_C \setminus \{q\}$ with transitions

$$(q, f_k, r_k)$$

Let

$$(q, g_q, q)$$

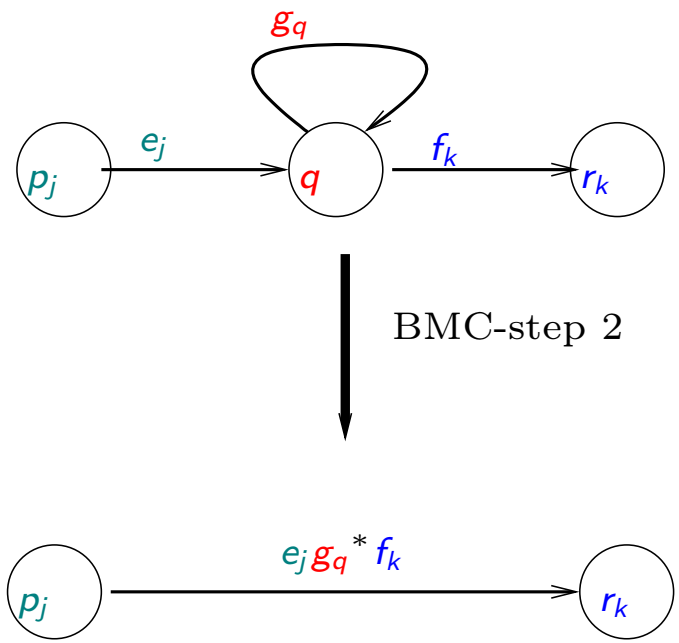
be the transition from q to q (we add te transition $(q, \emptyset, q)$ if there is no such transition).
We remove state q and add the transitions :

$$(p_j, e_j(g_q)^* f_k, r_k).$$

We obtain $\mathcal{C}' = \langle Q_C, \Sigma, \delta'_C, i, \{t\}\rangle$ which has one less state and still recognizes the same language.

BMC-algorithm : step 2

Step 2 :



Removing a state $q \notin \{i, t\}$.

BMC-algorithm : step 2

Step 2 :
We remove successively all the states in Q, until we obtain an efa of the form : $\mathcal{C} = \langle \{i, t\}, \Sigma, \delta_C, i, \{t\} \rangle$ where

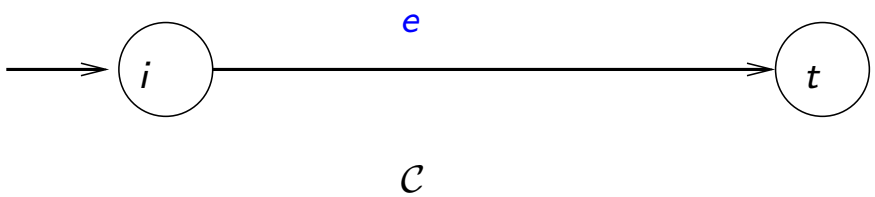
$$\delta_C = \{(i, e, t)\}.$$

Then :

$$\mathrm{L}(\mathcal{A}) = \mathrm{L}(\mathcal{C}) = \nu(e).$$

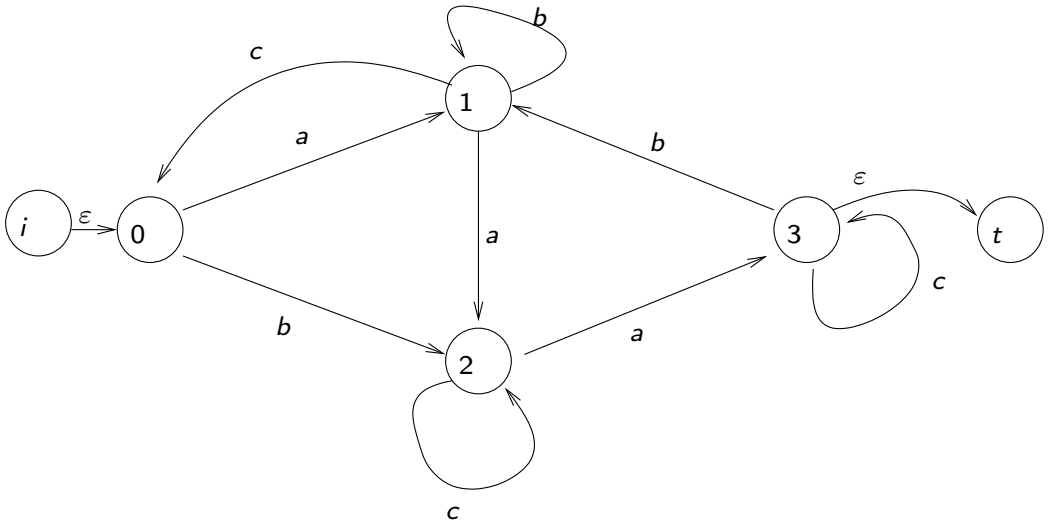
BMC-algorithm : step 3

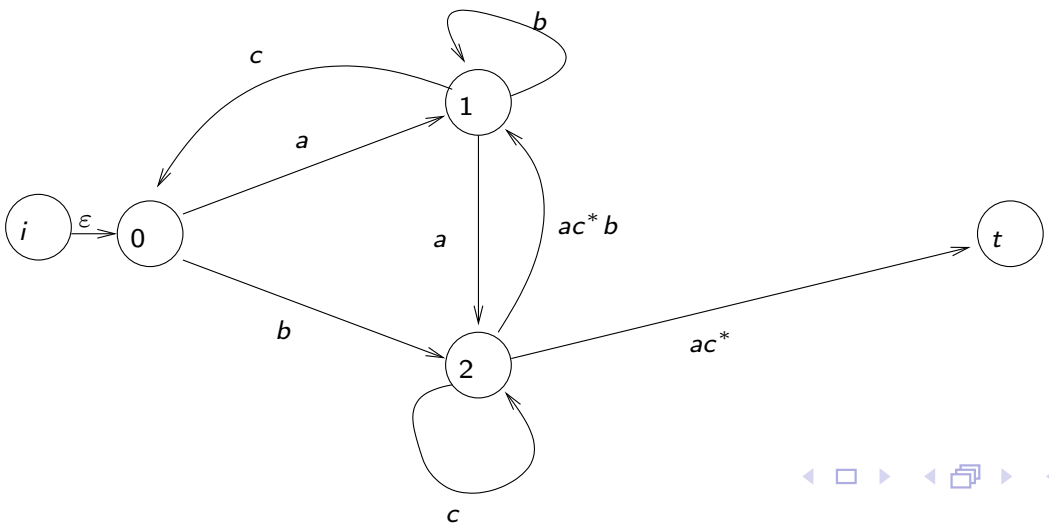
Step 3 :



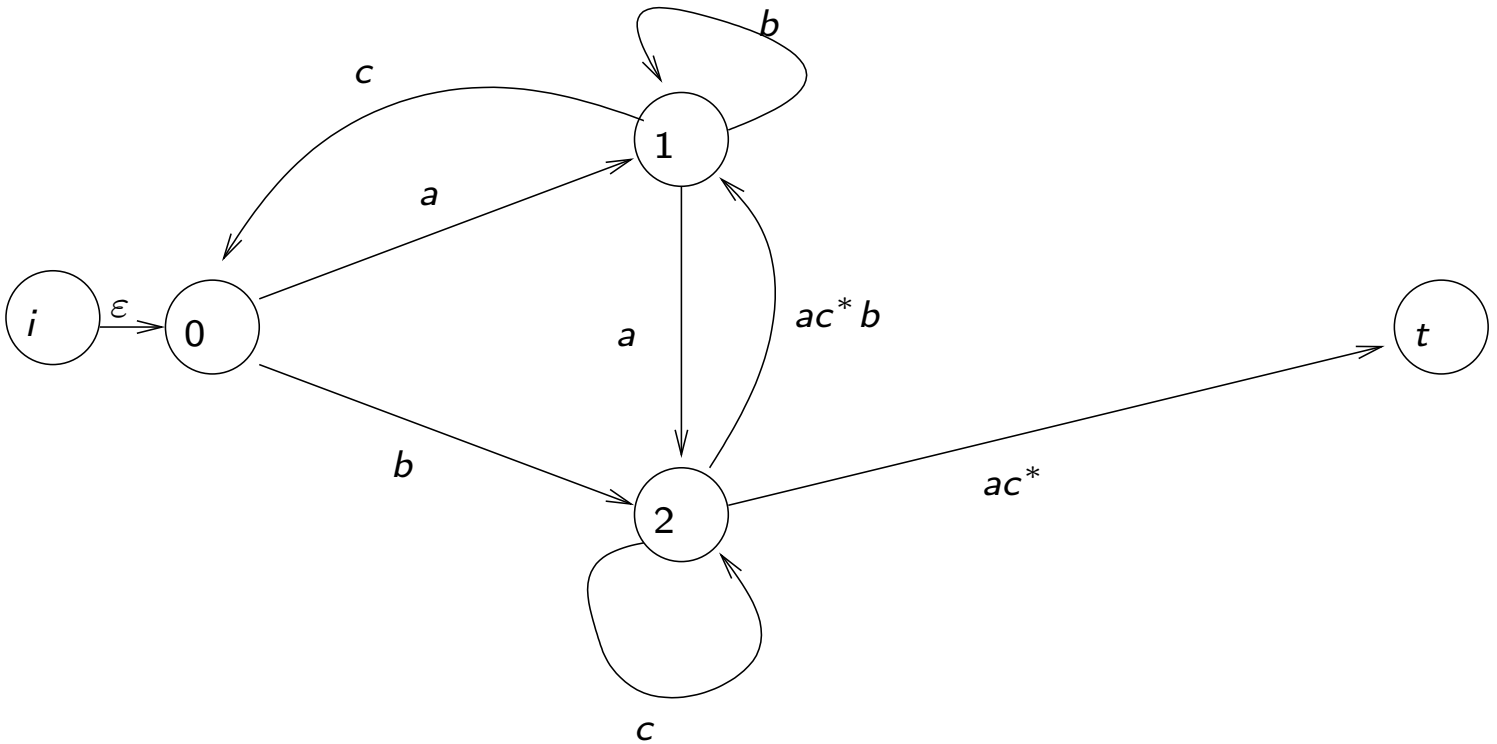
$\mathcal{C}$

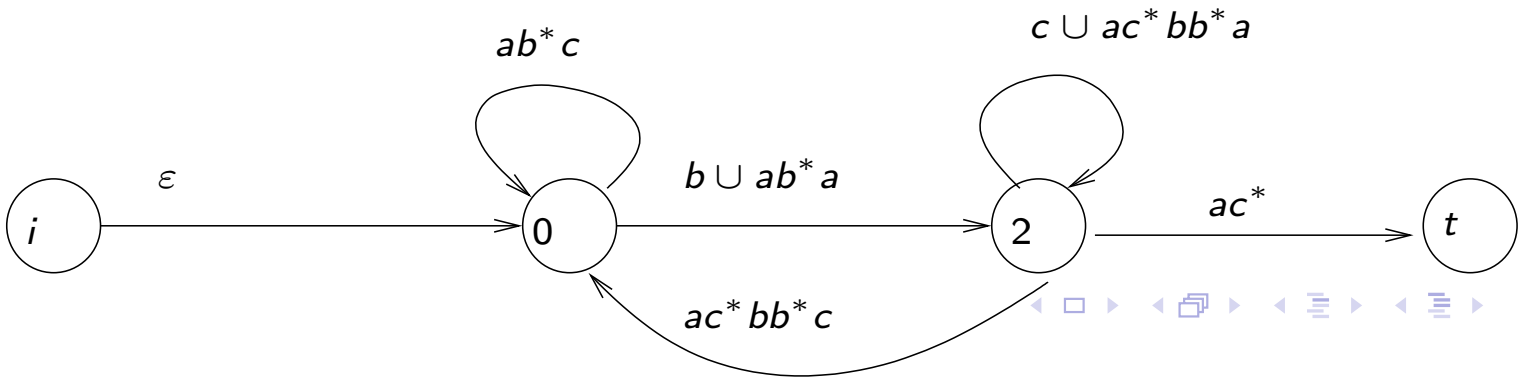
BMC-algorithm : example



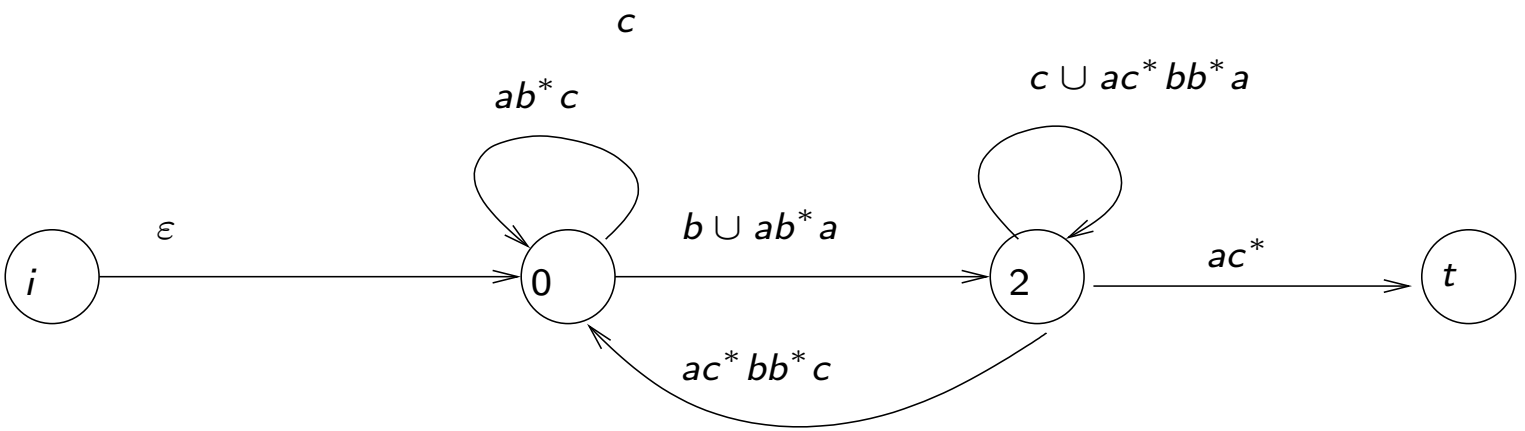


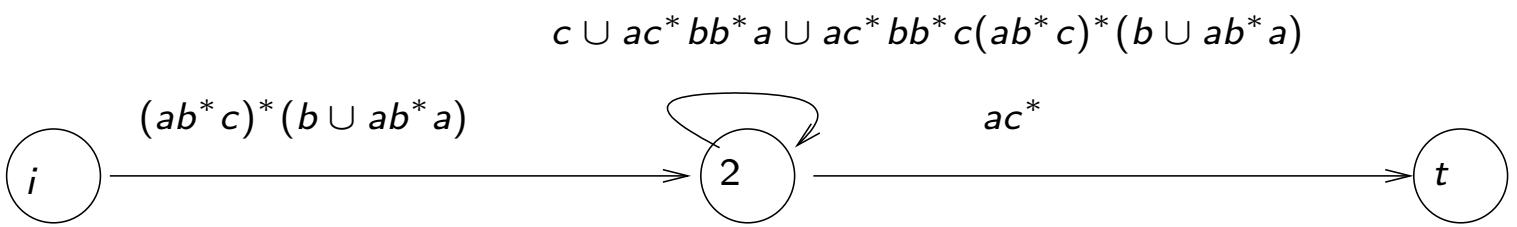
BMC-algorithm : example

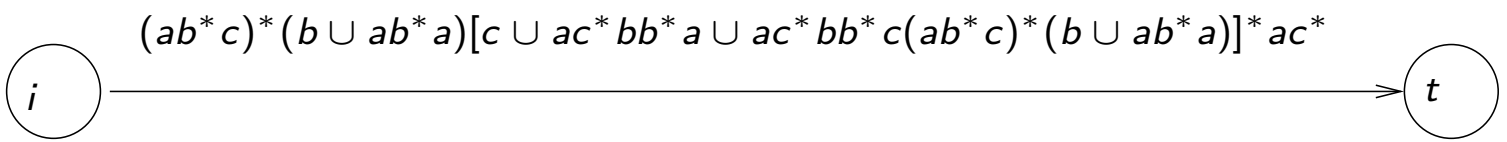




BMC-algorithm : example







Glushkov-algorithm

The Gluskov-algorithm : translates every regular expression e into an $\mathrm{nfa}\ \mathcal{A}$:

Three stages of generality :

stage 1 : translation of locally-testable languages.

stage 2 : translation of linear regular expressions.

stage 3 : translation of general regular expressions.

Locally-testable languages

A language $L \subseteq \Sigma^*$ is called locally testable if and only if there exist subsets $I, F \subseteq \Sigma$, $D \subseteq \Sigma^2$ such that

$$L = (I \cdot \Sigma^* \cap \Sigma^* \cdot F) \setminus \Sigma^* \bar{D} \Sigma^*$$

or

$$L = \{\varepsilon\} \cup [(I \cdot \Sigma^* \cap \Sigma^* \cdot F) \setminus \Sigma^* \bar{D} \Sigma^*]$$

where $\bar{D} = \Sigma^2 \setminus D$.
In words : L is the set of all words that begin with a letter in I, end with a letter in F and have all their factors of length 2 in the set D (the letter D (resp.I,F) are standing for "Digrams" (resp. "Initials","Final").

Glushkov-algo : stage 1

stage 1 : translation of a locally-testable language.

$$L = ((I \cdot \Sigma^*) \cap (\Sigma^* \cdot F)) \setminus (\Sigma^* \bar{D} \Sigma^*).$$

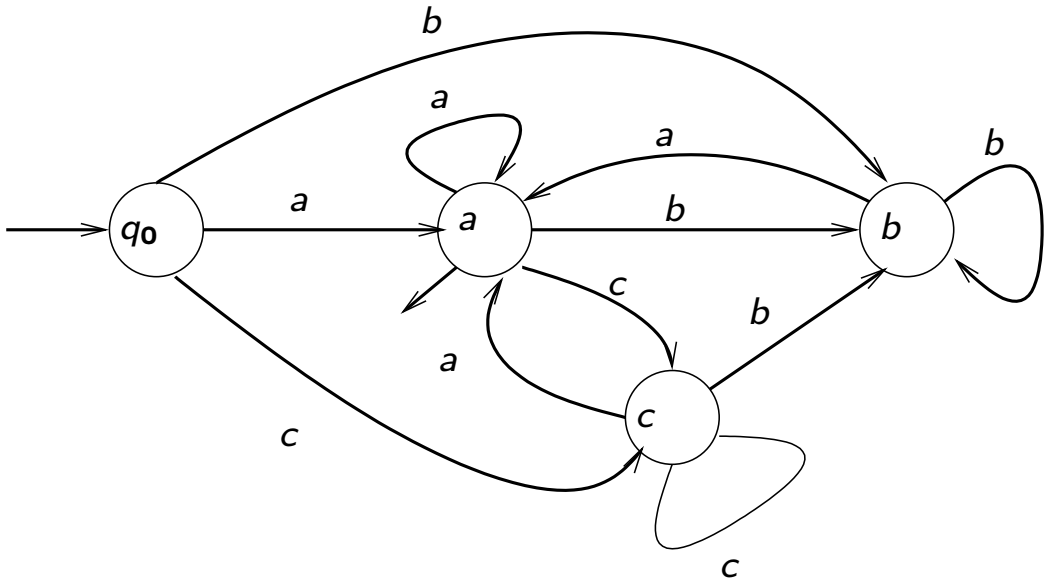
Let

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

be the nfa defined by :
$Q = \Sigma \cup \{q_0\}$ (where q_0 is a new symbol not in Σ)
$\delta = \{(q_0, x, x) \mid x \in I\} \cup \{(x, y, y) \mid x, y \in \Sigma, xy \in D\}$ Then

$$L = \mathrm{L}(\mathcal{A}).$$

Glushkov-algorithm : example



$$\Sigma = \{a, b, c\}, \quad L = (a\Sigma^* \cap \Sigma^* a) \setminus (\Sigma^* bc\Sigma^*)$$

Glushkov-algo : stage 1

stage 1 : translation of a locally-testable language.

$$L = \{\varepsilon\} \cup [((I \cdot \Sigma^*) \cap (\Sigma^* \cdot F)) \setminus (\Sigma^* \bar{D} \Sigma^*)].$$

Let

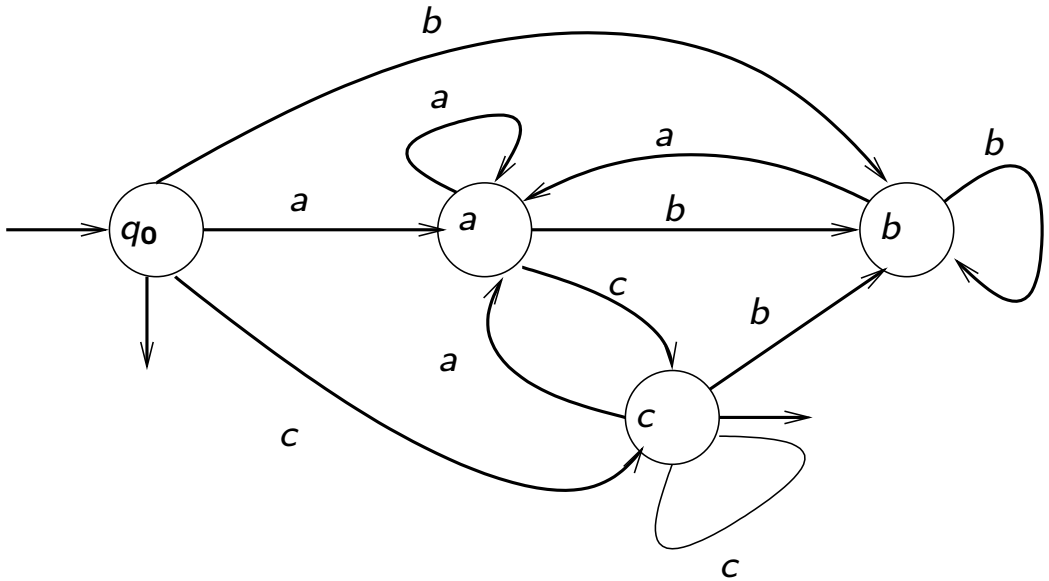
$$\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \cup \{q_0\} \rangle$$

be the nfa defined by :
$Q = \Sigma \cup \{q_0\}$ (where q_0 is a new symbol not in Σ)
$\delta = \{(q_0, x, x) \mid x \in I\} \cup \{(x, y, y) \mid x, y \in \Sigma, xy \in D\}$ Then

$$L = \mathrm{L}(\mathcal{A}).$$

NB : we have added q_0 in the set of final states.

Glushkov-algorithm : example



$$\Sigma = \{a, b, c\}, \quad L = \{\varepsilon\} \cup [(a\Sigma^* \cap \Sigma^* c) \setminus (\Sigma^* bc\Sigma^*)]$$

Glushkov-algo : stage 2

stage 2 : translation of a linear regular expression.
A regular expression e, over Σ , is called linear if each letter of Σ occurs at most once in the word e. Examples : $(((ab)^* \cup cd) \cup f)^*$ is linear
$(((ab)^* \cup ca) \cup f)^*$ is not linear

Proposition

If $e \in \mathrm{RE}(\Sigma)$ is linear then $\nu(e)$ is locally testable.

It suffices to prove that :
if $L_1 \in \mathrm{REG}(\Sigma_1^*), L_2 \in \mathrm{REG}(\Sigma_2^*)$ are locally testable languages over disjoint alphabets Σ_1, Σ_2, then :

$$L_1 \cup L_2, \quad L_1 \cdot L_2, \quad L_1^*$$

are locally-testable too.

Glushkov-algorithm : stage 2

stage 2 : translation of a linear regular expression.
For every language L over Σ, we define :

$$\mathrm{N}(L) = L \cap \{\varepsilon\},$$

$$\mathrm{I}(L) = \{x \in \Sigma \mid x^{-1}L \neq \emptyset\}, \quad \mathrm{F}(L) = \{x \in \Sigma \mid Lx^{-1} \neq \emptyset\},$$

$$\mathrm{D}(L) = \{xy \in \Sigma^2 \mid \exists \alpha, \beta \in \Sigma^*, \alpha xy\beta \in L\}.$$

Example : for $L = (abc)^* d(ba)$

$$\mathrm{N}(L) = \emptyset, \quad \mathrm{I}(L) = \{a, d\}, \mathrm{F}(L) = \{a\}, \mathrm{D}(L) = \{ab, bc, ca, cd, db, ba\}.$$

Glushkov-algorithm : stage 2

Remark : if L is locally testable i.e.

$$L = (I \cdot \Sigma^* \cap \Sigma^* \cdot F) \setminus \Sigma^* \bar{D} \Sigma^*$$

(possibly with the additional word ε) then

$$\mathrm{N}(L) = \emptyset, \quad \mathrm{I}(L) = I, \quad \mathrm{F}(L) = F, \quad D = \mathrm{D}(L), \quad \bar{D} = \Sigma^2 \setminus \mathrm{D}(L),$$

and $\mathrm{N}(L) = \{\varepsilon\}$ if $\varepsilon \in L$.
Hence, for a locally-testable language L, it suffices to compute $\mathrm{N}(L), \mathrm{I}(L), \mathrm{F}(L), \mathrm{D}(L)$ to be able to build a nfa recognizing L.

Glushkov-algorithm : stage 2

Given a regular expression e we define

$$\mathrm{N}(e) = \mathrm{N}(\nu(e)),\ \ \mathrm{I}(e) = \mathrm{I}(\nu(e)),\ \ \mathrm{F}(e) = \mathrm{F}(\nu(e)),\ \ \mathrm{D}(e) = \mathrm{D}(\nu(e)).$$

These four functions can be computed by the following recursive rules :

$f \in \mathrm{RE}(f)$	$\mathrm{N}(f)$	$\mathrm{I}(f)$	$\mathrm{F}(f)$
$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
ε	$\{\varepsilon\}$	$\emptyset$	$\emptyset$
$x \in \Sigma$	$\emptyset$	$\{x\}$	$\{x\}$
$e \cup e'$	$\mathrm{N}(e) \cup \mathrm{N}(e')$	$\mathrm{I}(e) \cup \mathrm{I}(e')$	$\mathrm{F}(e) \cup \mathrm{F}(e')$
$e \cdot e'$	$\mathrm{N}(e) \cap \mathrm{N}(e')$	$\mathrm{I}(e) \cup \mathrm{N}(e)\mathrm{I}(e')$	$\mathrm{F}(e)\mathrm{N}(e') \cup \mathrm{F}(e')$
e^*	$\{\varepsilon\}$	$\mathrm{I}(e)$	$\mathrm{F}(e)$
e^+	$\emptyset$	$\mathrm{I}(e)$	$\mathrm{F}(e)$

Glushkov-algorithm : stage 2

Recursive rules for $\mathrm{D}(*)$:

$f \in \mathrm{RE}(f)$	$\mathrm{D}(f)$
$\emptyset$	$\emptyset$
ε	$\emptyset$
$x \in \Sigma$	$\emptyset$
$e \cup e'$	$\mathrm{D}(e) \cup \mathrm{D}(e')$
$e \cdot e'$	$\mathrm{D}(e) \cup \mathrm{D}(e') \cup \mathrm{F}(e)\mathrm{I}(e')$
e^*	$\mathrm{D}(e) \cup \mathrm{F}(e)\mathrm{I}(e)$
e^+	$\mathrm{D}(e) \cup \mathrm{F}(e)\mathrm{I}(e)$

Glushkov-algorithm : stage 2

Stage 2 : Given a linear regular expression e :
- compute $\mathrm{N}(e), \mathrm{I}(e), \mathrm{F}(e), \mathrm{D}(e)$
- $\nu(e) = (I \cdot \Sigma^* \cap \Sigma^* \cdot F) \setminus \Sigma^* \bar{D} \Sigma^*$

for $I = \mathrm{I}(e), F = \mathrm{F}(e), D = \mathrm{D}(e)$.
- compute the nfa $\mathcal{A}$ associated to I, F, D
- add q_0 as final state if $\mathrm{N}(e) = \{\varepsilon\}$.

Glushkov-algorithm : stage 3

Given a general regular expression e over Σ :
Let $\Sigma' = \{x_i \mid 1 \leq i \leq |e|_x\}$. Let $e' \in \mathrm{RE}(\Sigma')$ be a linear regular expression that is mapped onto e by forgetting the indices in the symbols $(x_i) \in \Sigma'$.

Example

$$\Sigma = \{a, b, c\}, \quad e = (aba)^* b(bb(ab)^*)^* \cup (cba)^*.$$

then

$$\Sigma' = \{a_1, a_2, a_3, a_4, b_1, b_2, b_3, b_4, b_5, b_6, c_1\}$$

$$e' = (a_1 b_1 a_2)^* b_2(b_3 b_4(a_3 b_5)^*)^* \cup (c_1 b_6 a_4)^*.$$

We call e' a "linearization" of the regular expression e.

Glushkov-algorithm : stage 3

Stage 3 : Given a general regular expression e over Σ :
- linearize e into e'
- compute (by stage 2) a nfa $\mathcal{A}'$ such that $\nu(e') = \mathrm{L}(\mathcal{A}')$.
- let $\mathcal{A}$ be obtained from $\mathcal{A}'$ by replacing everywhere (i.e. in the input-alphabet and in the transitions) each letter x_i by x. Then

$$\nu(e) = \mathrm{L}(\mathcal{A}).$$

Glushkov-algorithm : example

$$e := (ab^*c)^*ab(a \cup b)^*$$

The linarization of e is :

$$e' := (a_1 b_1^* c_1)^* a_2 b_2 (a_3 \cup b_3)^*$$

We compute

$$\mathrm{I}(e') = \{a_1, a_2\}, \quad \mathrm{F}(L') = \{b_2, a_3, b_3\},$$

$\mathrm{D}(e') =$
$\{a_1 b_1, b_1 b_1, b_1 c_1, a_1 c_1, c_1 a_1, c_1 a_2, a_2 b_2, b_2 a_3, b_2 b_3, a_3 a_3, a_3 b_3, b_3 b_3, b_3 a_3\}.$

Glushkov-algorithm : example

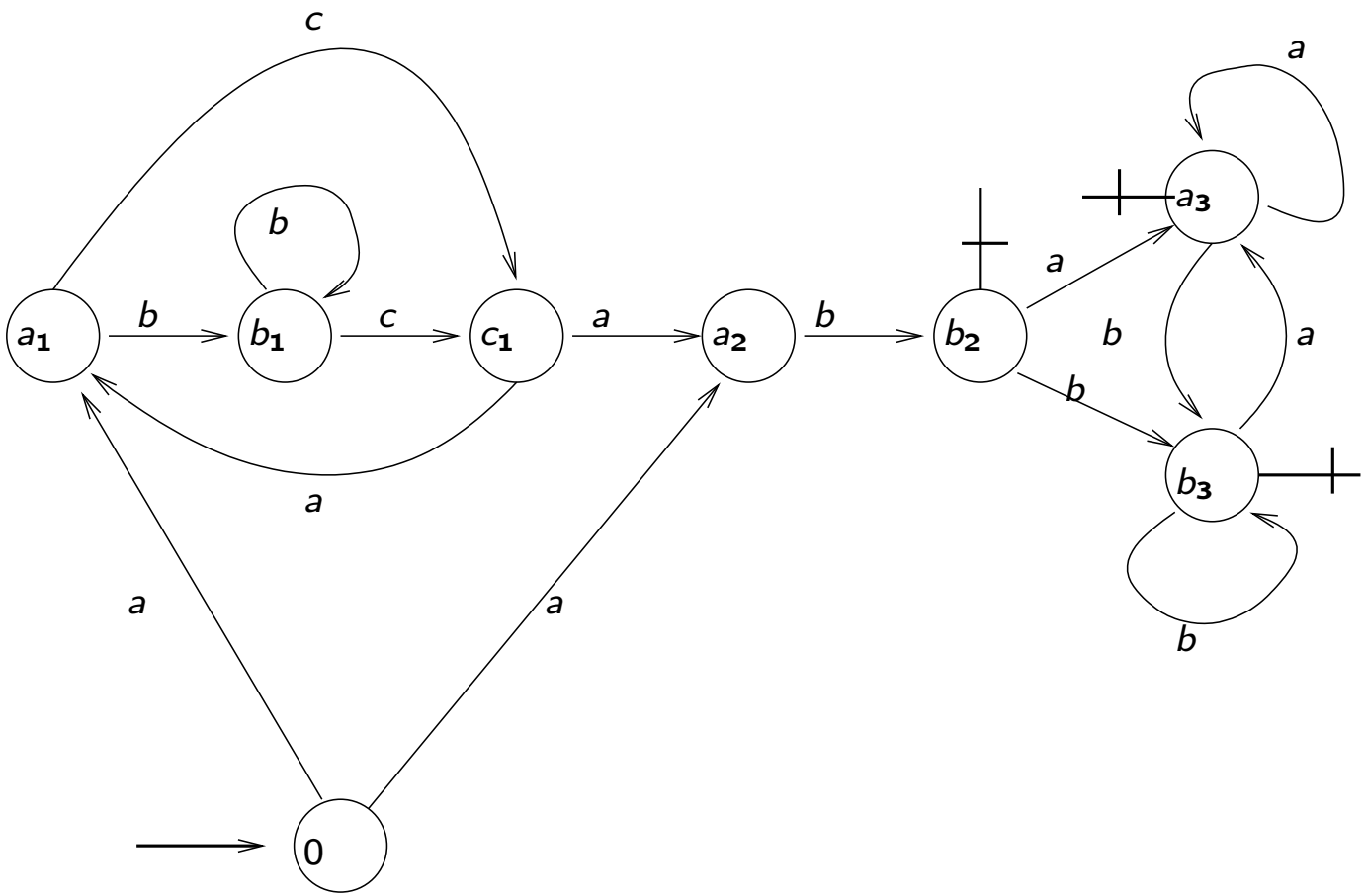


Figure – finite automaton for L_e

Kleene's theorem :summary

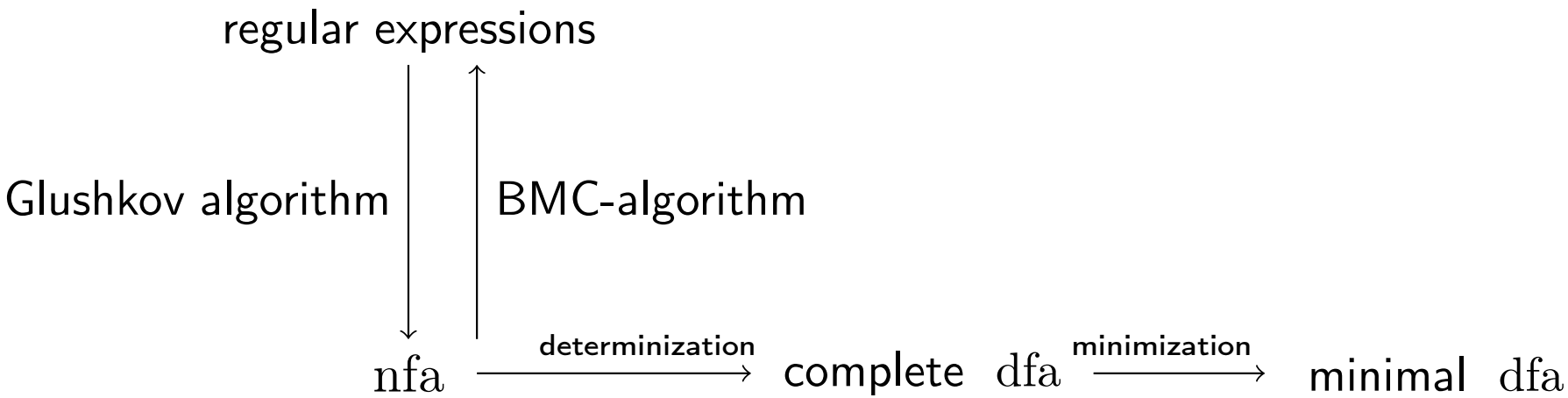


Figure – Kleene's theorem : summary.