

# M1 MPRI : Automates et Applications

## Course 1

### Basics of regular word languages

`kn@lri.fr`

January 16, 2024

- 1 Introduction
- 2 Alphabets, words and word operations
- 3 Languages and their operations
- 4 Regular languages
- 5 Deterministic automata
- 6 Non-deterministic automata
- 7 Minimization
- 8 Advanced concepts
- 9 Conclusions

Study automata and their connections with language theory and logic. Automata have two flavours:

- they are *algebraic* objects, defined mathematically and equipped with formal properties
- they are *computing* objects, you can run them to get a result

The course is an overview of different type of automata (and languages and logics), that give the vocabulary to explores such objects in more detailes in M2.

## Evaluation

- an assignment (that include proofs and a bit of programming) (40% of the final grade)
- a written exam, 2h, 60% of the final grade

Each session (tuesday morning) is divided (roughly) between 1h30 lecture and 1h30 exercises.

We will study the following concepts

- 1 Basics of regular word languages and word automata
- 2 Regular tree languages (1)
- 3 Regular tree languages (2)
- 4  $\omega$ -regular languages and Büchi automata
- 5 Automata and logics
- 6 (extra) Introduction to Markov chains (aka. automata with probabilities)

- 1 Introduction
- 2 Alphabets, words and word operations**
- 3 Languages and their operations
- 4 Regular languages
- 5 Deterministic automata
- 6 Non-deterministic automata
- 7 Minimization
- 8 Advanced concepts
- 9 Conclusions

## Définition (Alphabet)

*An alphabet is a set of elements called symbols or letters.*

We denote alphabets with  $\Sigma$  and  $a, b, \dots$  to range over symbols.

We always consider **finite alphabets** (even though most results can be generalized to infinite alphabets).

## Définition (Word)

A word over an alphabet  $\Sigma$  is a sequence of symbols. The empty word (made of 0 symbol) is written  $\epsilon$ .

We first focus on *finite* words, and we will later study the case of *infinite* words.

We write  $\Sigma^*$  the set of finite words over an alphabet  $\Sigma$  (we will come back to this notation and its meaning).

Words can be seen like a formal definition for *character strings* as used in programming languages. Alphabet correspond to the notion of *CharSet*, such as ASCII, Latin-1, Latin-9 or UCS (also known as “Unicode”).



## Définition

*Length* Let  $w \in \Sigma^*$  be an  $n$  symbols word. We call  $n$  the length of  $w$  and we write  $|w| = n$ .

Let  $\Sigma = \{a, b, c\}$ . The set of words of length at most two is over  $\Sigma$  is:

$$\{\epsilon, a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc\}$$

Consider a 64bit CPU. The set of possible values for a register is the set of words over  $\Sigma = \{0, 1\}$  of length 64 (exactly).

## Définition (Concatenation)

*Let  $u = a_1a_2 \dots a_n$  and  $v = b_1b_2 \dots b_m$  two words of length  $n$  et  $m$ . The concatenation of  $u$  and  $v$  is the word  $w = a_1a_2 \dots a_nb_1b_2 \dots b_m$  of length  $n + m$ . We write the concatenation by simple juxtaposition  $w = uv$  or  $u \cdot v$  (to disambiguate or emphasize the operation).*

# Properties of concatenation

Let  $\Sigma$  be an alphabet. The set  $(\Sigma^*, \cdot)$  equipped with the concatenation is a *monoid*:

Let  $\Sigma$  be an alphabet. The set  $(\Sigma^*, \cdot)$  equipped with the concatenation is a *monoid*:

**Identity element** for all word  $v \in \Sigma^*$ , we have  $\epsilon v = v \epsilon = v$

**Associativity** for all  $u, v, w \in \Sigma^*$ , we have  $u(vw) = (uv)w$ .

Remark : the length function  $|\_| : \Sigma^* \rightarrow \mathbb{N}$  is a monoid morphism between  $\Sigma^*$  and  $\mathbb{N}$ , equipped with addition.

- $|\epsilon| = 0$
- For all  $u, v \in \Sigma^*$ ,  $|u| + |v| = |uv|$

## Définition

*Power* Let  $v \in \Sigma^*$  and  $n \in \mathbb{N}$ , we define  $v^n$  by:

- $v^0 = \epsilon$
- $v^n = vv^{n-1}$ , for  $n \geq 1$

Example: let  $\Sigma = \{a, b\}$ , the word  $a^4b^2$  is *aaaabb*.

## Définition (Prefix, suffix)

Let  $w = uv$ , three words in  $\Sigma^*$ . We say that:

- $u$  is a prefix of  $w$
- $v$  is a suffix of  $w$

Question : Let  $\Sigma = \{a, b\}$ . Give all prefixes of the word *abbab*.

## Définition (Prefix, suffix)

Let  $w = uv$ , three words in  $\Sigma^*$ . We say that:

- $u$  is a prefix of  $w$
- $v$  is a suffix of  $w$

Question : Let  $\Sigma = \{a, b\}$ . Give all prefixes of the word *abbab*.

$\{\epsilon, a, ab, abb, abba, abbab\}$

## Définition (substring)

Let  $v$  a word in  $\Sigma^*$ . We say that  $u$  is a substring of  $v$  if there exists  $u_p$  and  $u_s$  such that  $v = u_p u u_s$ .

Warning in french: *substring*  $\equiv$  *facteur*



## Définition (embedding)

Let  $v = x_1 \dots x_n$  a word in  $\Sigma^*$ . The word  $u = x_{i_1} \dots x_{i_k}$  is an embedding of  $v$  if  $1 \leq i_1 < \dots < i_k \leq n$ .

Informally, an embedding of  $v$  is  $v$  where some symbols have been erased. For instance, for the word  $abcabc$ ,  $aa$ ,  $abab$  and  $cac$  are embedding.

Warning in french: *embedding*  $\equiv$  *sous-mot*.

## Définition (mirror)

Let  $v = x_1 \dots x_n$  a word in  $\Sigma^*$ , we call *mirror word* and write  $v^R$  the word  $v^R = x_n \dots x_1$ .

Remark: a word  $v$  such that  $v = v^R$  is called a *palindrome*.

- 1 Introduction
- 2 Alphabets, words and word operations
- 3 Languages and their operations**
- 4 Regular languages
- 5 Deterministic automata
- 6 Non-deterministic automata
- 7 Minimization
- 8 Advanced concepts
- 9 Conclusions

## Définition (language)

*A language  $L$  over an alphabet  $\Sigma$  is a subset of  $\Sigma^*$ . We write  $\mathcal{P}(\Sigma^*)$  the set of languages over  $\Sigma$ .*

Recall that  $\mathcal{P}(E)$  is the powerset of  $E$ , i.e. the set of all subsets of  $E$ . Do not confuse the following:  $\Sigma^*$ , the set of all words and  $\mathcal{P}(\Sigma^*)$  the set of all languages. If  $\Sigma = \{a, b\}$ , then:

- $\Sigma^* = \{\epsilon, a, b, aa, ab, ba, aaa, aab, aba, \dots\}$
- $\mathcal{P}(\Sigma^*) = \{\{\}, \{\epsilon\}, \{\epsilon, a\}, \{\epsilon, aaaaba, b\}, \dots\}$

Examples :

- $L = \{abc, bac, acb\}$  on  $\Sigma = \{a, b, c\}$
- $L = \{ab^n c \mid n \in \mathbb{N}\} = \{ac, abc, abbc, ab^3c, \dots\}$
- $L = \{a^n \mid n \text{ is prime}\} = \{aa, aaa, aaaaa, a^7, a^{11}, \dots\}$

Languages are *set*, and we can therefore talk about the usual set-theoretic operations on sets (union, intersection, complements). We can also define word specific operations.

## Définition (Union, intersection)

Let  $L_1$  et  $L_2$  two languages over **the same alphabet**  $\Sigma$ . We define the union language  $L_1 \cup L_2$  and the intersection  $L_1 \cap L_2$  with the usual set-theoretic operations.

## Définition (Complement)

Let  $L$  be a language over an alphabet  $\Sigma$ . We call complement of  $L$  and we write  $\bar{L}$  the set  $\Sigma^* \setminus L$ .

## Définition (concatenation of languages)

Let  $L_1$  and  $L_2$  be two languages over an alphabet  $\Sigma$ . The concatenation language  $L_1$  and  $L_2$  is the set  $L_1L_2$  defined as:

$$L_1L_2 = \{w \mid \exists u \in L_1, \exists v \in L_2, w = uv\}$$

## Définition (power of a language)

Let  $L$  be a language over an alphabet  $\Sigma$  and  $n \in \mathbb{N}$ . The power language  $L^n$  is defined as:

$$L^n = \{v_1 \dots v_n \mid v_i \in L \text{ for } 1 \leq i \leq n\}$$

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

Warning:  $L^n \neq \{v^n \mid v \in L\}$  (exercise).

## Définition (Kleene star)

Let  $L$  be a language over an alphabet  $\Sigma$ . The Kleene star of  $L$  is the language  $L^*$  defined as:

$$L^* = \bigcup_{n \in \mathbb{N}} L^n = \{v_1 \dots v_n \mid n \in \mathbb{N}, v_i \in L \text{ for } 1 \leq i \leq n\}$$

Exemple si  $L = \{aa, bb, ab\}$ , alors :

$$L^* = \{\epsilon, aa, bb, ab, aaaa, aabb, aaab, bbaa, bbbb, bbab, \dots\}$$

Stephen C. Kleene : American mathematician, student of Alonzo Church (like Alan Turing). Pioneered the theory of recursion and the study of computable functions.



## Définition (Mirror language)

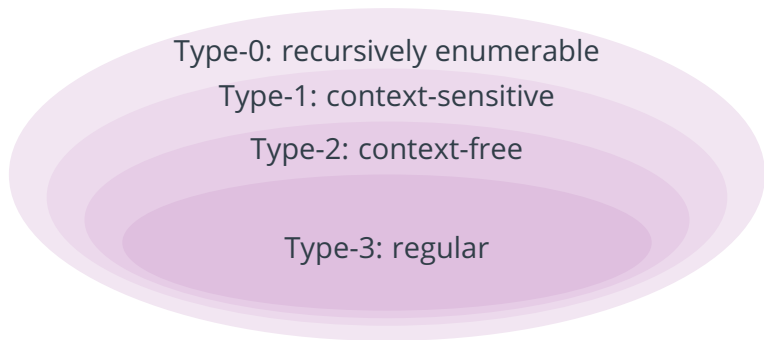
Let  $L$  be a language over  $\Sigma$ . We call mirror language of  $L$  and write  $L^R$  the language:

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

- 1 Introduction
- 2 Alphabets, words and word operations
- 3 Languages and their operations
- 4 Regular languages**
- 5 Deterministic automata
- 6 Non-deterministic automata
- 7 Minimization
- 8 Advanced concepts
- 9 Conclusions

# Motivation

A language is a set of words, which can be arbitrarily complex. As computer scientists, we are interested by *languages* for which we can decide whether a word is in it or not. Noam Chomsky (American linguist) established a hierarchy of formal languages:



It is known that inclusion is strict.

## Définition (Classe of rational languages)

Let  $\Sigma$  be an alphabet. Let  $Rat \subset \mathcal{P}(\Sigma^*)$  be the smallest set such that:

- $\emptyset \in Rat$  (empty language)
- $\forall x \in \Sigma, \{x\} \in Rat$  (singleton languages)
- si  $L \in Rat$ , alors  $L^* \in Rat$  (stability by Kleene star)
- si  $L_1, L_2 \in Rat$ ,  $L_1 \cup L_2 \in Rat$  (stability by union)
- si  $L_1, L_2 \in Rat$ ,  $L_1L_2 \in Rat$  (stability by concatenation)

We call  $Rat$  the class of rational languages. A language  $L$  is rational if and only if  $L \in Rat$ .

We build Rat as such (assume  $\Sigma = \{a, b\}$  for this example):

- $\text{Rat}_0 = \{\emptyset, \{a\}, \{b\}\}$
- $\text{Rat}_1 =$   
 $\text{Rat}_0 \cup \{\{a, b\}, \{ab\}, \{\epsilon\}, \{\epsilon, a, aa, aaa, \dots\}, \{\epsilon, b, bb, bbb, \dots\}\}$
- $\text{Rat}_2 = \text{Rat}_1 \cup \{a, ab\}, \{b, ab\}, \{\epsilon, a\}, \dots\}$
- ...

This (infinite) process has a limit  $\text{Rat}_\infty = \text{Rat}$ . The existence of such a limit is a consequence of Tarski's fixpoint theorem.

## Theorem (Tarski's fixed point)

*Let  $(L, \leq)$  be a complete lattice and  $f : L \rightarrow L$  a monotonic function over  $L$ . The set of fixedpoints of  $L$  also forms a complete lattice under  $\leq$ .*

- rational languages are closed under union, intersection, complement and mirror.
- the set of prefixes, suffixes, substring and embedding of a rational languages is rational.

## Définition (regular expression)

A regular expression (or regexp)  $r$  over an alphabet  $\Sigma$  is a finite production of the following grammar:

$r ::=$	$\emptyset$	empty set
	$\epsilon$	empty expression
	$x$	$\forall x \in \Sigma$ symbol
	$r^*$	Kleene star
	$r \mid r$	alternative
	$r r$	concatenation

Operator  $*$  is more binding than concatenation, itself more binding than the alternative. Thus, the regexp  $a|bc^*$  is parsed as  $a|(b(c^*))$ .



A regexp can be used to recognize a word, inductively on the structure of the expression:

- $\emptyset$  does not recognize any word
- the empty word is recognized by  $\epsilon$
- $a$  is recognized by  $a$
- a word  $v$  is recognized by  $r_1 \mid r_2$  if it is recognized by  $r_1$  or by  $r_2$
- a word  $v$  is recognized by  $r_1 r_2$  if there exists  $u$  and  $w$  such that  $v = uw$  and  $u$  is recognized by  $r_1$  and  $w$  is recognized by  $r_2$
- a word  $v$  is recognized by  $r^*$  if:
  - either  $v$  is empty
  - or  $v$  is recognized by  $r r^*$

# Exemple de reconnaissance

Consider the regexp  $a(b|c)^*$  and the word  $acb$ . The latter is recognized by the expression:

$acb : a \cdot ((b|c)^*)$  (concatenation)

$a : a$  (symbol)

$cb : (b|c)^*$  (Kleene star)

$c : (b|c)$  (alternative)

$c : c$  (symbol)

$b : (b|c)^*$  (Kleene star)

$b : (b|c)$  (alternative)

$b : b$  (symbol)

$\epsilon : (b|c)^*$  (Kleene star)

Let  $r$  be a regular expression. We can define  $L_r$  the set of all words recognized by  $r$ :

- $L_\emptyset = \emptyset$
- $L_\epsilon = \{\epsilon\}$
- $L_x = \{x\}, \forall x \in \Sigma$
- $L_{r_1|r_2} = L_{r_1} \cup L_{r_2}$
- $L_{r_1r_2} = L_{r_1}L_{r_2}$
- $L_{r^*} = (L_r)^*$

$\Rightarrow$  the language recognized by  $r$  is rational.

The set of rational languages is exactly the set of languages recognizable by a regular expression. We will henceforth use the more common term “regular language”.

Several regexp are useful in practice and can be encoded in our minimal subset:

- $r^?$   $\equiv r \mid \epsilon$  (repetition 0 or 1 time)
- $r^+$   $\equiv r r^*$  (repetition 1 time or more)
- $r^n$   $\equiv r r \dots r$  ( $n$  times)

- 1 Introduction
- 2 Alphabets, words and word operations
- 3 Languages and their operations
- 4 Regular languages
- 5 Deterministic automata**
- 6 Non-deterministic automata
- 7 Minimization
- 8 Advanced concepts
- 9 Conclusions

We have two equivalent kinds of object:

- Rational languages: the set of words having particular algebraic properties
- Regular expressions: a syntax and semantics to define sets of words in a compact manner

We would like to decide some problems or effectively compute the results of some operations.

- Membership: let  $w$  be a word, is  $w$  in  $L$  ?  
⇒ possible with a regexp but inefficient
- Let  $L$ , compute its complement
- Given two languages, compute their union (easy), intersection, concatenation ...
- Test whether a language recognize the empty word
- Test whether a language is finite
- Test whether a language is universal (is equal to  $\Sigma^*$ )
- Test whether two languages are equal
- Test if a language is a subset of another



## Définition (deterministic automaton)

A deterministic automaton (Deterministic Finite Automaton, DFA) is a 5-tuple  $(Q, \Sigma, \delta, q_0, F)$  where:

- $Q$  is a finite set of states
- $\Sigma$  is an alphabet
- $\delta : Q \times \Sigma \rightarrow Q$  is a transition function
- $q_0 \in Q$  is an initial state
- $F \subseteq Q$  is a set of accepting states

## Definition (Run)

Let  $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$  and  $v \in \Sigma^*$ . A *run* (or an execution) of  $\mathcal{A}$  for  $v = x_1 \dots x_n$  is a sequence of states  $r_0, \dots, r_n$  in  $Q$  such that:

- $r_0 = q_0$  (initial state)
- $r_{i+1} = \delta(r_i, x_{i+1}), \forall 0 \leq i \leq n$

if  $r_n \in F$ , we say that the run is *accepting*.

We say that  $\mathcal{A}$  stops its run in state  $r_n$ .

## Définition (DFA recognizability)

Let  $v$  un mot de  $\Sigma^*$  et  $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$  a DFA. We say that  $\mathcal{A}$  recognizes  $v$  sif and only if there exists an accepting run for  $v$  by  $\mathcal{A}$ .

We say that  $\mathcal{A}$  accepts  $v$  or recognizes  $v$ . If there is no such run, we say that  $\mathcal{A}$  rejects  $v$  (or does not accept).

# Example

Consider the word *abba* and the DFA  $\mathcal{A}$ :

$$\mathcal{A} = (\{q_0, q_1, q_2, q_\perp\}, \{a, b\}, \delta, q_0, \{q_2\})$$

$$\delta(q_0, a) = q_1 \quad \delta(q_2, a) = q_\perp$$

$$\delta(q_0, b) = q_\perp \quad \delta(q_2, b) = q_\perp$$

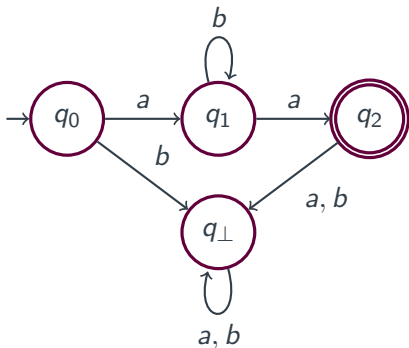
$$\delta(q_1, a) = q_2 \quad \delta(q_\perp, a) = q_\perp$$

$$\delta(q_1, b) = q_1 \quad \delta(q_\perp, b) = q_\perp$$

The word *abba* is recognized  $\mathcal{A}$ , with the run  $q_0, q_1, q_1, q_2$

# Graphical representation

We can represent  $\delta$  graphically as a direct graph. The vertices are the states and the edges are labeled with symbols of  $\Sigma$ :



## Définition (Language of an automaton)

Let  $\mathcal{A}$  be a DFA. The language  $L_{\mathcal{A}}$  is the set of all words  $\Sigma^*$  recognized by  $\mathcal{A}$ .

La classe of *reconnizable* languages is the set of all languages that can be recognized by some automaton.

## Définition (Complete automaton)

A DFA,  $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$  is complete if and only if  $\delta$  is a total function.

## Lemme (Completion)

For all automaton  $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ , there exists a complete automaton  $\mathcal{A}'$  such that  $L_{\mathcal{A}} = L_{\mathcal{A}'}$ .

Proof : if  $\mathcal{A}$  is not complete, let  $\mathcal{A}' = (Q \cup \{q_{\perp}\}, \Sigma, \delta', q_0, F)$  and:

$$\delta'(q, x) = \begin{cases} \delta(q, x) & \text{if defined} \\ q_{\perp} & \text{otherwise} \end{cases}$$
$$\delta'(q_{\perp}, x) = q_{\perp}, \quad \forall x \in \Sigma$$

A state  $q_{\perp}$  such that  $\forall x \in \Sigma, \delta(q_{\perp}, x) = q_{\perp}$  is called a *sink state*.

Incomplete DFA have are more practical since we don't have to store useless transitions (i.e. transitions that don't help recognize a word).

If there isn't any possible transition, the run is stuck therefore the word is rejected.

We can always assume that we work with complete automata (some theorems require it), but we will often draw *incomplete* ones to save space.



Let  $\mathcal{A}_1$  and  $\mathcal{A}_2$  be two automata. We can construct  $\mathcal{A}_{1\wedge 2}$  such that  $L_{\mathcal{A}_{1\wedge 2}} = L_{\mathcal{A}_1} \cap L_{\mathcal{A}_2}$ . Similarly, we can construct  $\mathcal{A}_{1\vee 2}$  such that  $L_{\mathcal{A}_{1\vee 2}} = L_{\mathcal{A}_1} \cup L_{\mathcal{A}_2}$ .

Let  $\mathcal{A}_1 = (Q_1, \Sigma, \delta_1, q_0, F_1)$  and  $\mathcal{A}_2 = (Q_2, \Sigma, \delta_2, p_0, F_2)$ . Define  $\mathcal{A}_{1 \wedge 2} = (Q_{1 \wedge 2}, \Sigma, \delta_{1 \wedge 2}, (q_0, p_0), F_{1 \wedge 2})$  with:

- $Q_{1 \wedge 2} = Q_1 \times Q_2$  (each state is a pair of states of  $Q_1$  and  $Q_2$ )
- $\forall (q, p) \in Q_{1 \wedge 2}, \forall x \in \Sigma, \delta_{1 \wedge 2}((q, p), x) = (\delta_1(q, x), \delta_2(p, x))$  (we assume that the automata are complete).
- $F_{1 \wedge 2} = \{(q, p) \mid q \in F_1 \wedge p \in F_2\}$

The automaton  $\mathcal{A}_{1 \wedge 2}$  recognizes  $L_{\mathcal{A}_1} \cap L_{\mathcal{A}_2}$

- Assume that  $v$  is recognized by  $\mathcal{A}_{1\wedge 2}$ . By definition, there exists an accepting run  $(r_0, s_0) \dots (r_n, s_n)$  (avec  $n = |v|$ ). Consider the sequence  $r_0 \dots r_n$ . By induction on  $n$ , it is an accepting run of  $\mathcal{A}_1$  for  $v$ , and therefore  $v \in L_{\mathcal{A}_1}$ . Likewise,  $v \in L_{\mathcal{A}_2}$  therefore  $L_{\mathcal{A}_{1\wedge 2}} \subseteq L_{\mathcal{A}_1} \cap L_{\mathcal{A}_2}$ .
- Assume that  $v \in L_{\mathcal{A}_1} \cap L_{\mathcal{A}_2}$ . Since  $v \in L_{\mathcal{A}_1}$ , there exists an accepting Run  $r_0 \dots r_n$ . Likewise  $v \in L_{\mathcal{A}_2}$ , thus there exists an accepting run  $s_0 \dots s_n$ . Run  $(r_0, s_0) \dots (r_n, s_n)$  of  $\mathcal{A}_{1\wedge 2}$  is accepting:
  - $(r_0, s_0) = (q_0, p_0)$
  - $(r_{i+1}, s_{i+1}) = (\delta(r_i, x_{i+1}), \delta(s_i, x_{i+1}))$  by construction.
  - $r_n \in F_1, s_n \in F_2$  therefore  $(r_n, s_n) \in F_{1\wedge 2}$ . □

Construction similar to  $\mathcal{A}_{1 \wedge 2}$ . We just change the definition of accepting states:  $\mathcal{A}_{1 \vee 2} = (Q_{1 \vee 2}, \Sigma, \delta_{1 \vee 2}, (q_0, p_0), F_{1 \vee 2})$  with:

- $Q_{1 \vee 2} = Q_1 \times Q_2$
- $\forall (q, p) \in Q_{1 \vee 2}, \forall x \in \Sigma, \delta_{1 \vee 2}((q, p), x) = (\delta_1(q, x), \delta_2(p, x))$
- $F_{1 \vee 2} = \{(q, p) \mid q \in F_1 \vee p \in F_2\}$

Indeed we only require that a word is recognized either by  $\mathcal{A}_1$  or by  $\mathcal{A}_2$ .

The product construction can be seen as an automaton that simulates both automata in parallel. For each symbol, the product automaton takes a transition in each of the two source automata. The construction is simplified by considering complete automata. Otherwise:

- For the intersection, we can stop if one of the two automaton cannot make a transition
- For the union we must *continue* even if we are stuck on one of the automata.

The definition remains almost the same  $\delta_{1\wedge 2}$  (we have to limit ourselves to pairs of existing states in both automata) but is cumbersome for pour  $\delta_{1\vee 2}$ , since a state is either a pair or a single state.

Let  $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ . The complement automataon of  $\mathcal{A}$ ,  $\bar{\mathcal{A}}$  is defined as:

$$\bar{\mathcal{A}} = (Q, \Sigma, \delta, q_0, Q \setminus F)$$

We have  $\overline{L_{\mathcal{A}}} = L_{\bar{\mathcal{A}}}$  (trivial). Again, the construction considers complete autoamta (otherwise it does not work).

An alternative version to compute the union is to use De Morgan's laws:

$$\mathcal{A}_1 \vee \mathcal{A}_2 = \overline{\overline{\mathcal{A}_1} \cap \overline{\mathcal{A}_2}}$$

- 1 Introduction
- 2 Alphabets, words and word operations
- 3 Languages and their operations
- 4 Regular languages
- 5 Deterministic automata
- 6 Non-deterministic automata**
- 7 Minimization
- 8 Advanced concepts
- 9 Conclusions

As we have seen, given a word  $v$ , a complete DFA is always know its unique destination state, for the next letter in  $v$ .

We can generalize DFA to non deterministic-automata (NFA). For a given state, and a given symbol, the transition function returns *a set* of possible states. If *one of these states* end in an accepting run the word is accepted.

Intuitively, we can see this in several ways:

- The NFA “guesses” the next correct state (oracle)
- The NFA tries every states in sequence until it finds one leading to an accepting run (*backtracking*)
- The NFA tries all possible states in parallel



## Définition (Non deterministic automaton)

A Non-deterministic automaton (Non-deterministic Finite Automaton, NFA) is a 5-tuple  $(Q, \Sigma, \delta, I, F)$  where:

- $Q$  is a finite set of states
- $\Sigma$  is an alphabet
- $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$  is a transition function
- $I \subseteq Q$  is a set of initial states
- $F \subseteq Q$  is a set of accepting states

## Definition (Exécution)

Let  $\mathcal{A} = (Q, \Sigma, \delta, I, F)$  and  $v \in \Sigma^*$ . A run of  $\mathcal{A}$  for  $v = x_1 \dots x_n$  is a sequence of states  $r_0, \dots, r_n \in Q$  such that:

- $r_0 \in I$  (initial state)
- $r_{i+1} \in \delta(r_i, x_{i+1}), \forall 0 \leq i \leq n$

If  $r_n \in F$  we say that the run is accepting.

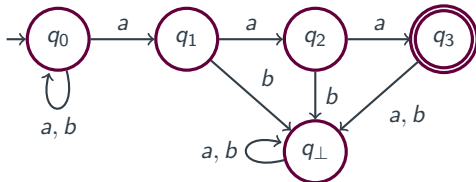
Contrary to DFA, NFA may have several runs for the same word and even several accepting runs.

## Définition (Reconnaissance par un NFA)

Let  $v$  a word  $\Sigma^*$  and  $\mathcal{A} = (Q, \Sigma, \delta, I, F)$  a NFA. We say that  $\mathcal{A}$  recognizes  $v$  if and only if there exists an accepting run for  $v$  by  $\mathcal{A}$ .

# Example

f Consider:  $\mathcal{A} = (\{q_0, q_1, q_2, q_3, q_\perp\}, \{a, b\}, \delta, \{q_0\}, \{q_3\})$



For the word  $abaaa$ , there are two possible runs:

- $q_0, q_1, q_\perp, q_\perp, q_\perp, q_\perp$
- $q_0, q_0, q_1, q_2, q_3$  which is accepting.

Remark : this automaton recognizes all words ending with three  $a$ . When the automaton reads an  $a$  it has to “guess” whether it stays in  $q_0$  (if there are  $b$  further in the word) or if it goes to  $q_1$  (for the third  $a$  to the end).

A NFA  $\mathcal{A} = (Q, \Sigma, \delta, I, F)$  is complete if and only if:

- $\delta$  is total
- $\forall q \in Q, \forall x \in \Sigma, \delta(q, x) \neq \emptyset$

As for DFA, any NFA can be completed with a sink state.

## Théorème (Determinisation)

Let  $\mathcal{A} = (Q, \Sigma, \delta, I, F)$  be a NFA. There exists

$\mathcal{A}_{det} = (Q_{det}, \Sigma, \delta_{det}, q_0, F_{det})$ , a DFA, such that  $L_{\mathcal{A}} = L_{\mathcal{A}_{det}}$ .

We give a constructive proof.

The DFA simulates “in parallel” all the possible runs of the NFA. Each state of the DFA represents the set of states in which the NFA can be.

Consider the function  $\delta_{\text{det}} : \mathcal{P}_f(Q) \times \Sigma \rightarrow \mathcal{P}_f(Q)$  defined as:

$$\delta_{\text{det}}(P, x) = \{q \mid q \in \delta(p, x) \text{ for } p \in P\}$$

Consider now the sequence of states:

- $S_0 = \{I\}$
- $S_{i+1} = S_i \cup \bigcup_{P \in S_i} \bigcup_{x \in \Sigma} \{\delta_{\text{det}}(P, x)\}$

Then, there exists  $l$  such that  $S_{l+1} = S_l$  (in other words, we can saturate  $S_i$  until we reach the limit  $S_l$ ).

## Proof (2)

This limit exists since  $S_i$  is strictly increasing (at each step we add the previous set) and the set of states we can add is finite, it's  $\mathcal{P}_f(Q)$  (of size  $2^{|Q|}$ ). The DFA is:

$$\mathcal{A}_{\text{det}} = (S_I, \Sigma, \delta_{\text{det}}, I, \{P \in S_n \mid P \cap F \neq \emptyset\})$$

Indeed, for each word  $v \in \Sigma^*$  :

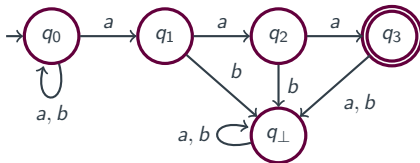
- Si  $R_0 \dots R_n$  is an accepting run  $\mathcal{A}_{\text{det}}$  for  $v$ , then there exists  $\exists r_0 \in R_0 = I, \exists r_1 \in R_1 \dots r_n \in R_n$  such that  $r_0 \dots r_n$  is an accepting run of  $v$  for  $\mathcal{A}$ .
- If  $r_0 \dots r_n$  is one accepting run of  $\mathcal{A}$  for  $v$ , alors  $\exists R_0 = I \in S_I, \dots, \exists R_n \in S_I$  such that  $\forall i, r_i \in R_i$  et  $R_0 \dots R_n$  is an accepting run for  $v$  of  $\mathcal{A}_{\text{det}}$ .

Each direction is shown by induction on  $n$ , the length of the word  $v$ .



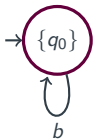
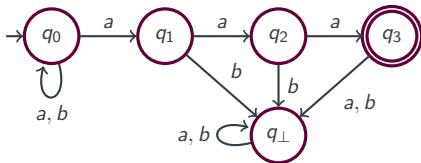
# Example

Consider:  $\mathcal{A} = (\{q_0, q_1, q_2, q_3, q_\perp\}, \{a, b\}, \delta, \{q_0\}, \{q_3\})$



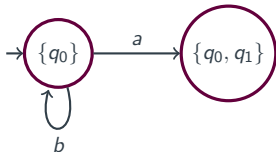
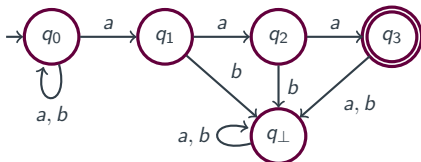
# Example

Consider:  $\mathcal{A} = (\{q_0, q_1, q_2, q_3, q_\perp\}, \{a, b\}, \delta, \{q_0\}, \{q_3\})$



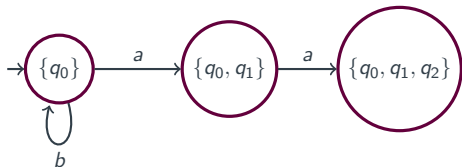
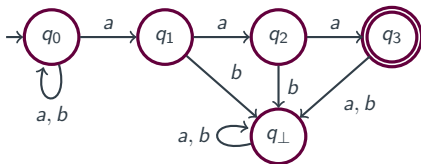
# Example

Consider:  $\mathcal{A} = (\{q_0, q_1, q_2, q_3, q_\perp\}, \{a, b\}, \delta, \{q_0\}, \{q_3\})$



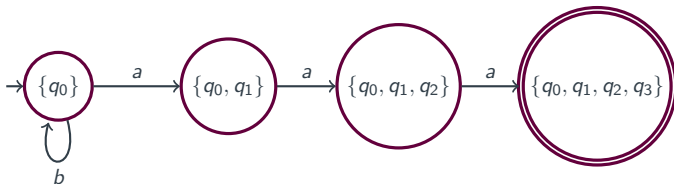
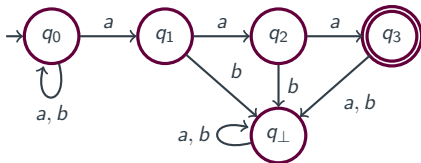
# Example

Consider:  $\mathcal{A} = (\{q_0, q_1, q_2, q_3, q_\perp\}, \{a, b\}, \delta, \{q_0\}, \{q_3\})$



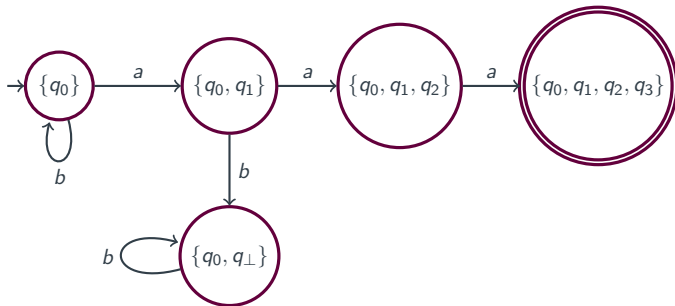
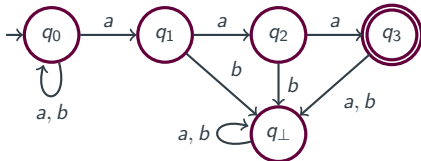
# Example

Consider:  $\mathcal{A} = (\{q_0, q_1, q_2, q_3, q_\perp\}, \{a, b\}, \delta, \{q_0\}, \{q_3\})$



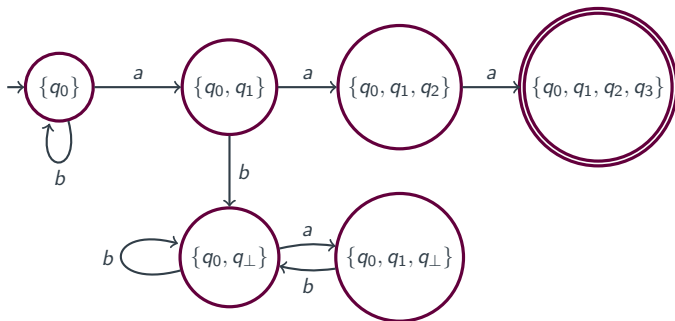
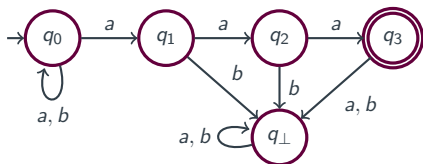
# Example

Consider:  $\mathcal{A} = (\{q_0, q_1, q_2, q_3, q_\perp\}, \{a, b\}, \delta, \{q_0\}, \{q_3\})$



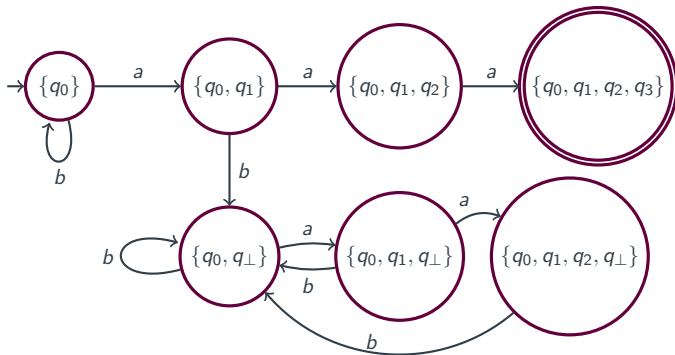
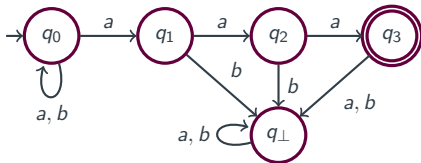
# Example

Consider:  $\mathcal{A} = (\{q_0, q_1, q_2, q_3, q_\perp\}, \{a, b\}, \delta, \{q_0\}, \{q_3\})$



# Example

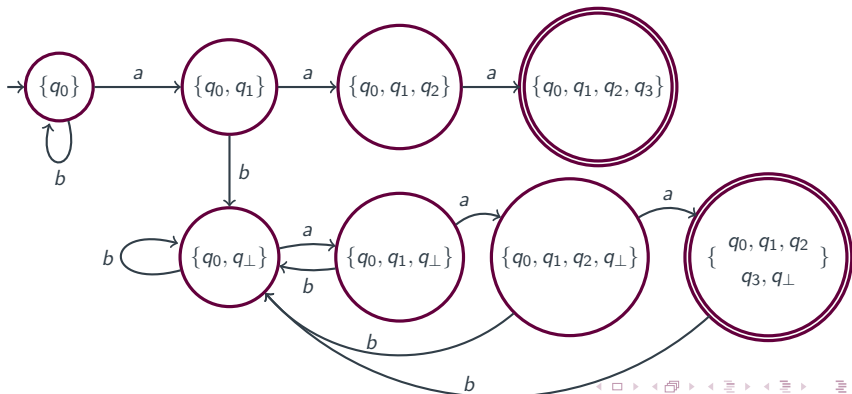
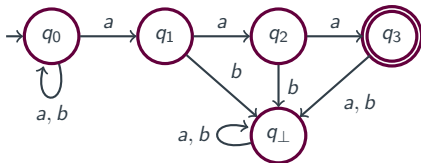
Consider:  $\mathcal{A} = (\{q_0, q_1, q_2, q_3, q_\perp\}, \{a, b\}, \delta, \{q_0\}, \{q_3\})$





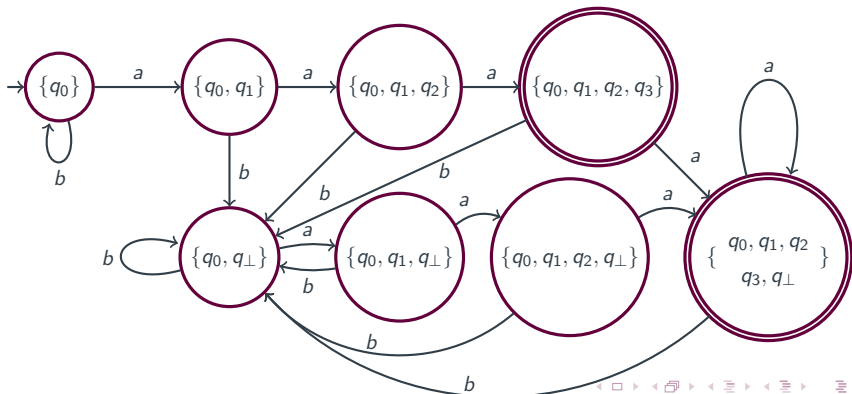
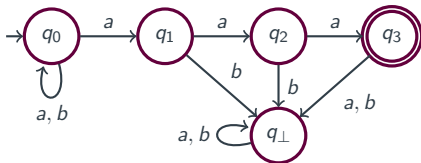
# Example

Consider:  $\mathcal{A} = (\{q_0, q_1, q_2, q_3, q_\perp\}, \{a, b\}, \delta, \{q_0\}, \{q_3\})$



# Example

Consider:  $\mathcal{A} = (\{q_0, q_1, q_2, q_3, q_\perp\}, \{a, b\}, \delta, \{q_0\}, \{q_3\})$



As we have seen, being recognizable by a DFA is equivalent to being recognized by a NFA :

- $Rec(DFA) \subseteq Rec(NFA)$  : Trivial (a DFA is a particular case of an NFA where each state of the transition function is a singleton)
- $Rec(NFA) \subseteq Rec(DFA)$  : By the determinization theorem.

Since DFA and NFA are equivalent, we expect that that NFA have the same closure properties. It is true, but one needs to be careful.

Let  $\mathcal{A}_1 = (Q_1, \Sigma, \delta_1, l_1, F_1)$  and  $\mathcal{A}_2 = (Q_2, \Sigma, \delta_2, l_2, F_2)$  be two NFA. The automaton  $\mathcal{A}_{1 \vee 2} = (Q_1 \cup Q_2, \Sigma, \delta_{1 \vee 2}, l_1 \cup l_2, F_1 \cup F_2)$  recognizes the language  $L_{\mathcal{A}_1} \cup L_{\mathcal{A}_2}$ , with  $\delta_{1 \vee 2}$  defined by:

$$\delta_{1 \vee 2}(q, x) = \begin{cases} \delta_1(q, x) & \text{si } q \in Q_1 \\ \delta_2(q, x) & \text{si } q \in Q_2 \end{cases}$$

It's an efficient computation :  $O(|Q_1| + |\delta_1| + |Q_2| + |\delta_2|)$

We can use the product construction for two NFA. Let  $\mathcal{A}_1 = (Q_1, \Sigma, \delta_1, l_1, F_1)$  et  $\mathcal{A}_2 = (Q_2, \Sigma, \delta_2, l_2, F_2)$  be two NFA. The product automaton  $\mathcal{A}_{1 \wedge 2} = (Q_1 \times Q_2, \Sigma, \delta_{1 \wedge 2}, l_1 \times l_2, F_1 \times F_2)$  recognizes the language  $L_{\mathcal{A}_1} \cap L_{\mathcal{A}_2}$ , with  $\delta_{1 \wedge 2}$  est défini par :

$$\delta_{1 \wedge 2}((r, s), x) = \delta_1(r, x) \times \delta_2(s, x)$$

The straightforward technique that reverses accepting and non accepting states does **not** work for NFA. Indeed, for a DFA:

- If a run is accepting for  $v$  in the DFA, the *same run* in the complement DFA is rejecting (and vice versa).

For NFA, if we switch accepting and non accepting states:

- If a run for a word  $v$  is accepting in the NFA, this same run in the (buggy) complement is rejecting. But maybe there are other non-accepting run, and by definition those become accepting, which is wrong!

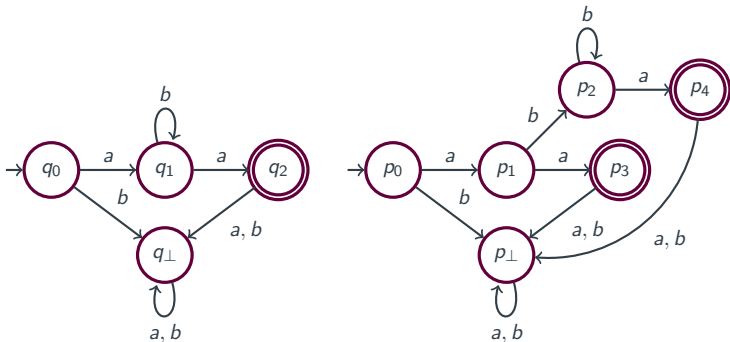
⇒ the only solution is to determinize and take the complement, which may cause an exponential blow-up in the number of states.

- 1 Introduction
- 2 Alphabets, words and word operations
- 3 Languages and their operations
- 4 Regular languages
- 5 Deterministic automata
- 6 Non-deterministic automata
- 7 Minimization**
- 8 Advanced concepts
- 9 Conclusions



# Minimal automaton

Remark that two automata (here two DFA) can recognize the same language:



Intuitively, the left-hand side “behaves like”  $ab^*a$ , while the right-hand side “behaves like”  $a(a|b^+a)$ .

## Théorème (Myhill-Nerode theorem)

Let  $L$  be a language. Let  $w \in \Sigma^*$  be a word. We define the set  $\text{Next}_L(w) = \{t \in \Sigma^* \mid wt \in L\}$ . Let  $\equiv_L$  be the equivalence relation defined by  $u \equiv_L v$  if and only if  $\text{Next}_L(u) = \text{Next}_L(v)$ .

- $L$  is recognizable if and only if the number of equivalence classes of  $\equiv_L$  is finite. Call  $k$  this number.
- There exists a unique DFA with  $k$  states recognizing  $L$ , and there are no DFA with a smaller number  $n < k$  of states recognizing  $L$ .

We just give the intuition and the minimization algorithm.

- $\text{Next}_L(w) = \{t \in \Sigma^* \mid wt \in L\}$  is the set of suffixes that allows one to complete  $w$  such that it is in  $L$ . For instance, for the finite language  $L = \{aab, bab, aaa, baa\}$ ,  $\text{Next}_L(a) = \{ab, aa\}$ .
- Two words  $u$  and  $v$  are equivalent ( $u \equiv_L v$ ) if and only if  $\text{Next}_L(u) = \text{Next}_L(v)$ . For our example:  $a \equiv_L b$ . This means that if we consider  $u$  and  $v$  as start of words recognized by the automaton, their suffixes can be recognized by the same part of the automaton.

# Moore's Algorithm (minimization)

**Input** a DFA  $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$

**Output** a DFA  $\mathcal{A}_{\min} = (Q_{\min}, \Sigma, \delta_{\min}, q'_0, F_{\min})$  où  $|Q_{\min}| \leq |Q|$

$$\mathbb{P} = \{F, Q \setminus F\}$$

repeat:

$$\mathbb{P}' := \mathbb{P}$$

for all  $C \in \mathbb{P}$ :

for all  $x \in \Sigma$ :

$$C := \emptyset$$

for all  $D \in \mathbb{P}$ :

$$C := C \cup \{q \in C \mid \delta(q, x) \in D\}$$

$$\mathbb{P} := \mathbb{P} \setminus \{C\} \cup C$$

while  $\mathbb{P} \neq \mathbb{P}'$

# Moore's Algorithm (minimization) (2)

Return  $(\mathbb{P}, \Sigma, \delta_{\min}, \{q_0\}, \{F\})$  where  $\delta_{\min}$  is defined by:

$$\delta_{\min}(C, x) = [\delta(q, x)]_{\mathbb{P}}, \text{ en choisissant } q \in C$$

$[r]_{\mathbb{P}}$  is the unique set  $C$  of  $\mathbb{P}$  containing  $r$ .

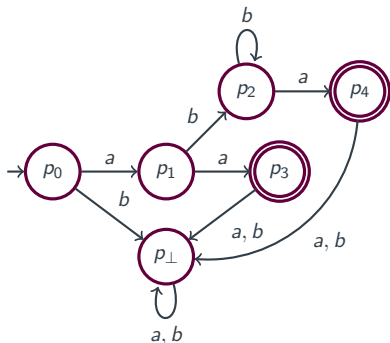
The algorithm builds approximations of the equivalence classes, that is puts in the same sets of states the state that recognize the same suffix. At the end, each such set is a state of the minimal automaton, all redundant states have been merged.

The algorithm starts with two rough sets : accepting (recognize  $\epsilon$ ) and non accepting states (do not recognize  $\epsilon$  as a suffix).

For each set  $C$ , for each symbol, we partition  $C$  into the subsets that lead to the same set of states.

# Exemple

We minimize:

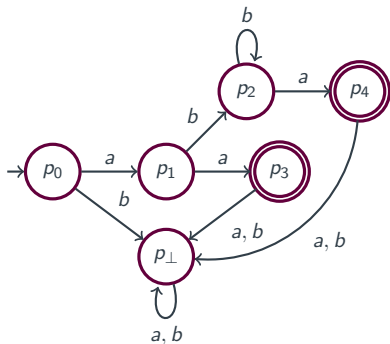


$$\mathbb{P}_0 \mid \{ \{p_0, p_1, p_2, p_{\perp}\}, \{p_3, p_4\} \}$$

accepting vs. non-accepting

# Exemple

We minimize:



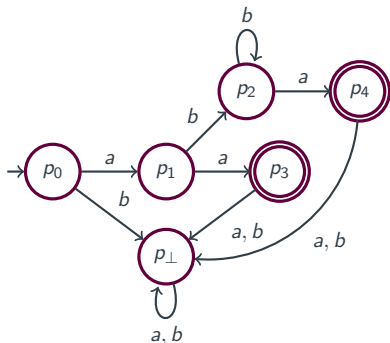
$\mathbb{P}_0$	$\{\{p_0, p_1, p_2, p_\perp\}, \{p_3, p_4\}\}$
$\mathbb{P}_1$	$\{\{p_0, p_\perp\}, \{p_1, p_2\}, \{p_3, p_4\}\}$

accepting vs. non-accepting  
a distinguish  $p_1, p_2$  and  $p_0, p_\perp$



# Exemple

We minimize:

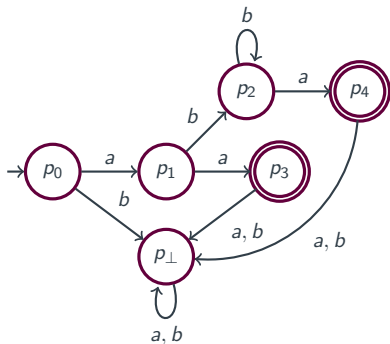


$$\begin{array}{l|l} \mathbb{P}_0 & \{\{p_0, p_1, p_2, p_\perp\}, \{p_3, p_4\}\} \\ \mathbb{P}_1 & \{\{p_0, p_\perp\}, \{p_1, p_2\}, \{p_3, p_4\}\} \\ \mathbb{P}_2 & \{\{p_0\}, \{p_\perp\}, \{p_1, p_2\}, \{p_3, p_4\}\} \end{array}$$

accepting vs. non-accepting  
 $a$  distinguish  $p_1, p_2$  and  $p_0, p_\perp$   
 $a$  distinguish  $p_0$  and  $p_\perp$

# Exemple

We minimize:

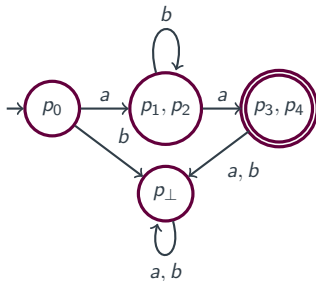
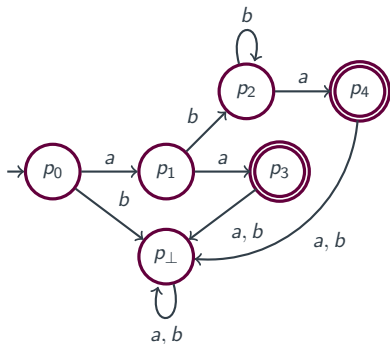


$\mathbb{P}_0$	$\{\{p_0, p_1, p_2, p_\perp\}, \{p_3, p_4\}\}$
$\mathbb{P}_1$	$\{\{p_0, p_\perp\}, \{p_1, p_2\}, \{p_3, p_4\}\}$
$\mathbb{P}_2$	$\{\{p_0\}, \{p_\perp\}, \{p_1, p_2\}, \{p_3, p_4\}\}$
$\mathbb{P}_3$	no change

accepting vs. non-accepting  
a distinguish  $p_1, p_2$  and  $p_0, p_\perp$   
a distinguish  $p_0$  and  $p_\perp$

# Exemple

We minimize:



$\mathbb{P}_0$	$\{\{p_0, p_1, p_2, p_\perp\}, \{p_3, p_4\}\}$
$\mathbb{P}_1$	$\{\{p_0, p_\perp\}, \{p_1, p_2\}, \{p_3, p_4\}\}$
$\mathbb{P}_2$	$\{\{p_0\}, \{p_\perp\}, \{p_1, p_2\}, \{p_3, p_4\}\}$
$\mathbb{P}_3$	no change

accepting vs. non-accepting  
a distinguish  $p_1, p_2$  and  $p_0, p_\perp$   
a distinguish  $p_0$  and  $p_\perp$

- 1 Introduction
- 2 Alphabets, words and word operations
- 3 Languages and their operations
- 4 Regular languages
- 5 Deterministic automata
- 6 Non-deterministic automata
- 7 Minimization
- 8 Advanced concepts**
- 9 Conclusions

We can generalize NFA with  $\epsilon$ NFA which can go from one state to another without reading any symbol from the input.

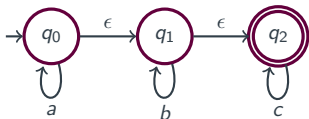
## Définition (NFA with empty moves)

An automaton with empty moves ( $\epsilon$ NFA) is a 5-tuple  $(Q, \Sigma, \delta, I, F)$  where:

- $Q$  is a finite set of states états
- $\Sigma$  is an alphabet
- $\delta : Q \times \Sigma \cup \{\epsilon\} \rightarrow \mathcal{P}_f(Q)$  is a transition function
- $I \subseteq Q$  is a set of initial states
- $F \subseteq Q$  is a set of accepting states

# Example

Consider the  $\epsilon$ NFA below. One needs to be careful to define the notion of run:



From the initial state  $q_0$ , the word  $cc$  is accepted. Indeed, we can go from  $q_0$  to  $q_1$  (without consuming a symbol), then from  $q_1$  to  $q_2$ , and then loop twice on  $q_2$ . We have a run  $q_0, q_1, q_2, q_2$  (of size 4) while the word is of size 2. We need to consider  $\epsilon$  as a “symbol”.

## Definition (Exécution)

Let  $\mathcal{A} = (Q, \Sigma, \delta, I, F)$  and  $v \in \Sigma^*$ . A *run* of  $\mathcal{A}$  for  $v$  is a sequence of states  $r_0, \dots, r_n \in Q$  such that there exists a sequence of words  $v = v_1 \cdot \dots \cdot v_n$ ,  $v_i \in \Sigma \cup \{\epsilon\}$ , such that:

- $r_0 \in I$  (état initial)
- $r_{i+1} \in \delta(r_i, v_{i+1})$ ,  $\forall 0 \leq i \leq n$

In our previous example, the word  $cc$  was recognized as  $\epsilon\epsilon cc$ .

Consider an  $\epsilon$ NFA with a transition function  $\delta$ . We call the  $\epsilon$ -closure of a state  $q$  the set of states  $E(q)$  as the limit of:

- $E_0(q) = \{q\}$
- $E_i(q) = E_{i-1} \cup \bigcup_{q' \in E_{i-1}} \delta(q', \epsilon)$

Given a set of states  $S$ , we write  $E(S) = \{E(q) \mid q \in S\}$

This gives all the states reachable by following only empty moves.



## Théorème (Removing empty moves)

For all  $\epsilon$ NFA,  $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ , there exists a NFA

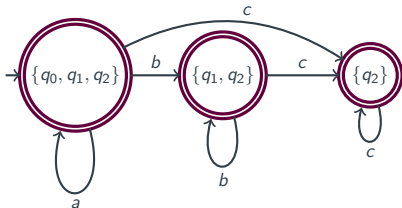
$\mathcal{A}_N = (Q_N, \Sigma, \delta_N, I_N, F_N)$  such that  $L_{\mathcal{A}} = L_{\mathcal{A}_N}$ .

# Proof

We construct:

- $Q_N = \{E(q) \mid q \in Q\}$ , each states of  $\mathcal{A}_N$  is a set of states reachable by empty moves.
- $I_N = \{E(q) \mid q \in I\}$  et  $F_N = \{P \mid P \in Q_N, P \cap F \neq \emptyset\}$ .
- $\delta_N(P, x) = \bigcup_{p \in E(P)} \delta(p, x)$

For our example:



Remark : the construction can make a state  $q$  such as:



unreachable, we have to remove them afterwards.

# Why bother with $\epsilon$ NFA ?

Why do we introduce  $\epsilon$ NFA if they are equivalent to NFA ?

# Why bother with $\epsilon$ NFA ?

Why do we introduce  $\epsilon$ NFA if they are equivalent to NFA ?

They allow to build *inductively* an  $\epsilon$ NFA from a regular expression.

# Why bother with $\epsilon$ NFA ?



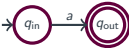
Why do we introduce  $\epsilon$ NFA if they are equivalent to NFA ?

They allow to build *inductively* an  $\epsilon$ NFA from a regular expression.

This construction was given by Ken Thompson, father of C and Unix.

This construction takes as input a regular expression and build an  $\epsilon$ NFA recognizing the same language.

Every automaton generated by the construction have a unique initial state  $q_{in}$  and a unique accepting state  $q_{out}$  :

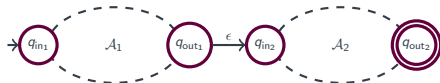
- $\emptyset$  (empty expression) : 
- $\epsilon$  (accept  $\epsilon$ ) : 
- $a$  (accepts one symbol) : 

These are all the basic cases.

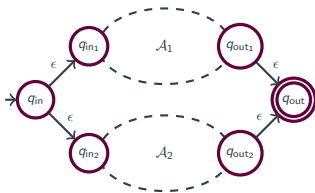
# Thompson's construction (2)

Let  $r_1$  and  $r_2$  be two regexp with respective automata  $\mathcal{A}_1$  and  $\mathcal{A}_2$  :

■  $r_1 \cdot r_2$  (concatenation)



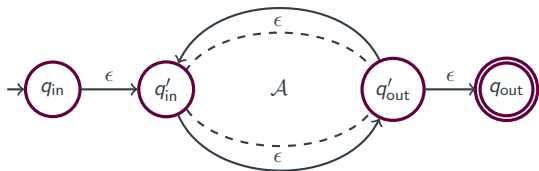
■  $r_1 \mid r_2$  (alternative)



Warning, in  $\mathcal{A}_1$  and  $\mathcal{A}_2$   $q_{out1}$  et  $q_{out2}$  are accepting, but they may not be in the resulting automaton.

# Thompson's construction (3)

For the Kleene star, let  $r$  be a regexp and  $\mathcal{A}$  its automaton:



A more complex construction is given by the Glushkov-Berry-Sethi algorithm, which builds directly a NFA from  $r$ . It was shown that removing the empty moves from Thompson automaton gives the Glushkov-Berry-Sethi automaton.



Twofold :

- It shows that  $\text{Reg} \subseteq \text{Rec}$ , since for each regexp we can build an automaton
- It gives an efficient way to compute membership

Twofold :

- It shows that  $\text{Reg} \subseteq \text{Rec}$ , since for each regexp we can build an automaton
- It gives an efficient way to compute membership

What about the other direction?

There are several algorithm to transform an automaton into a regular expression. The one we give is the algorithm of Brzozowski and McCluskey, dubbed *state reduction algorithm*. It uses, as an intermediate representation, *generalized automata*.

## Définition (Generalized automaton)

A Generalized automaton (GFA) is a 5-tuple  $(Q, \Sigma, \delta, q_{in}, q_{out})$  where:

- $Q$  is a finite set of states
- $\Sigma$  is an alphabet
- $\delta : Q \times \text{Reg}(\Sigma) \rightarrow \mathcal{P}_f(Q)$  is a transition function
- $I \subseteq Q$  is a set of initial states
- $F \subseteq Q$  is a set of accepting states

$\text{Reg}(\Sigma)$  is the set of regular expressions over  $\Sigma$ . These automata are labeled by regexp (and not symbols) and have a unique initial state and a unique accepting state.

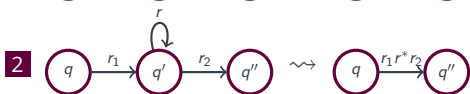
# State reduction algorithm

Given an NFA, we consider it as a GFA whose transitions are labeled by either a symbol or  $\epsilon$ . If needed, we can add a single initial and accepting states and link them using empty moves.

We apply two rules:

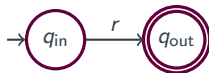
merging of transitions: 

state reduction : 2 cases:



# State reduction algorithm

At each step, we reduce the number of states or we reduce the number of transitions. The algorithm terminates when the automaton is reduced to:

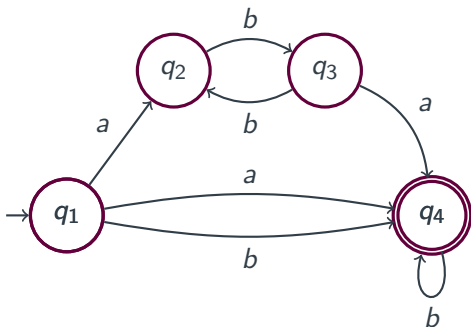


where  $r$  is a regular expression recognizing the same languages as the initial NFA. Warning, the resulting regexp may be complex and exponentially larger than the NFA. In particular, if we apply Thompson's construction to a regexp and state reduction afterwards, we obtain an equivalent regexp but **not** the initial one in general.

It shows that  $\text{Rec} \subseteq \text{Reg}$  et donc que  $\text{Rec} = \text{Reg} = \text{Rat}$ . This equivalence is called Kleene's Theorem. Its proof is the double inclusion, using the constructions we have seen (or other equivalent constructions).

# Example

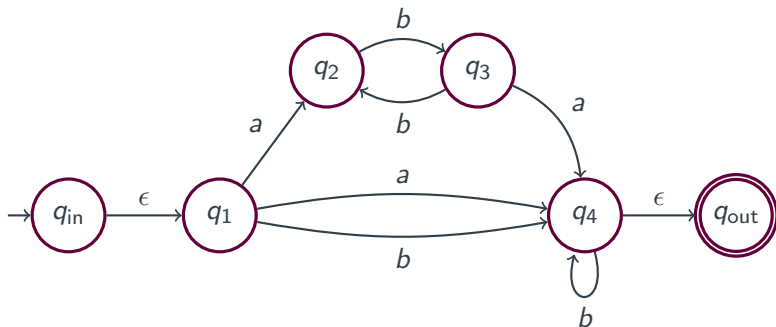
Consider the *NFA*:





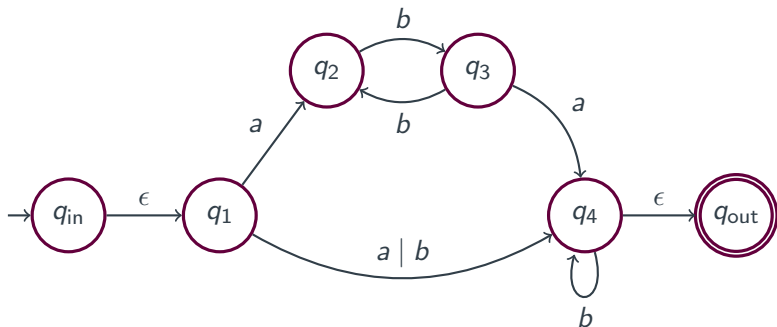
# Example

Consider the *NFA*:



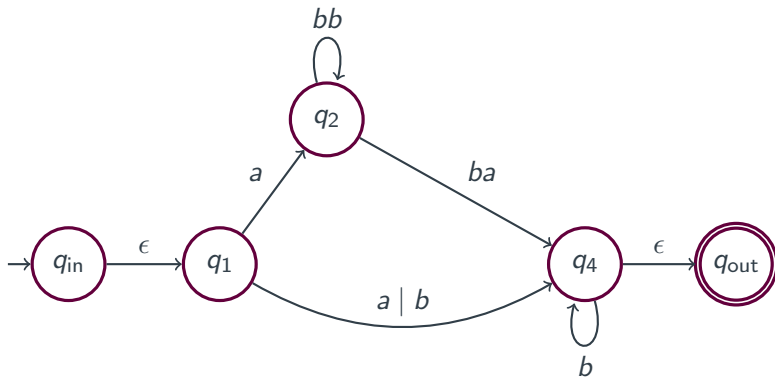
# Example

Consider the *NFA*:



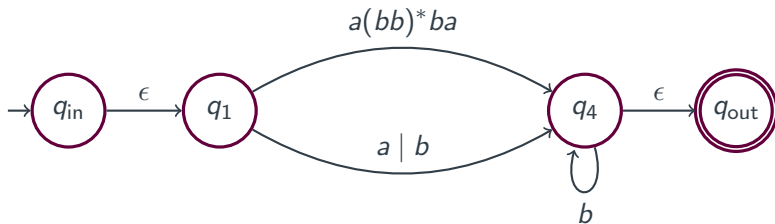
# Example

Consider the *NFA*:



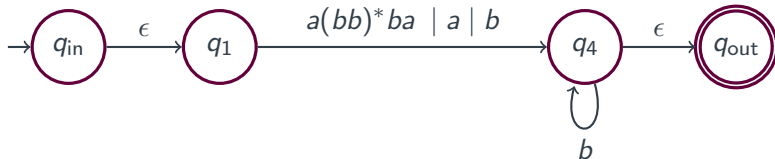
# Example

Consider the *NFA*:



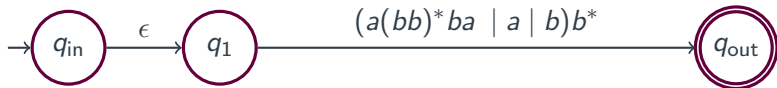
# Example

Consider the *NFA*:



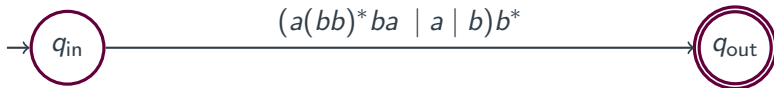
# Example

Consider the *NFA*:



# Example

Consider the *NFA*:



Given a regular language, we have algorithms to decide or compute:

- the union, intersection, complement, mirror (exercise) of a language (using automata)
- emptiness of the language (the minimal DFA is reduced to a single, non accepting state)
- universality of a language (its complement is empty)
- $L_A \subseteq L_B$



Given a regular language, we have algorithms to decide or compute:

- the union, intersection, complement, mirror (exercise) of a language (using automata)
- emptiness of the language (the minimal DFA is reduced to a single, non accepting state)
- universality of a language (its complement is empty)
- $L_A \subseteq L_B \Leftrightarrow L_A \setminus L_B = \emptyset \Leftrightarrow L_A \cap \overline{L_B} = \emptyset$
- $L_A = L_B$  (either double inclusion, or determinize, minimize and compare the automata)

What is missing ?

# Testing whether a language is regular ?

We won't get an algorithm :(

## Théorème

*Testing whether a context-free language is regular is undecidable.*

We will have to settle for something less powerfull.

We can give a criteria to prove that a language is *not* regular.

## Lemme (Pumping lemma)

Let  $L$  be a regular language over an alphabet  $\Sigma$ . There exists  $p \geq 1$  such that for all word  $w \in L$  such that  $|w| \geq p$ , there exists  $x, y, z \in \Sigma^*$  such that:

- $w = xyz$
- $y \neq \epsilon$
- $|xy| \leq p$
- $\forall n \geq 0, xy^n z \in L$

# Pumping lemma (2)

What does it mean ?

# Pumping lemma (2)

What does it mean ? For every regular language, there is a length of word  $p$  such that, if the language recognizes words longer than  $p$ , then there *must* be a “loop” in the automaton or a Kleene star in the regular expression. Therefore, there is a substring of the word (the one below the star or in the loop), which can be pumped, that is repeated arbitrarily many times while staying in the language.

# Pumping lemma (3)

How do we use it ?

# Pumping lemma (3)

How do we use it ? It's a *necessary condition* for a language to be regular, therefore:

- ⇒ if a language is regular, it can be “pumped”. By contrapositive, if a language cannot be “pumped” then it is not regular.
- ⇐ if a language can be pumped, it is not necessarily regular.

# Pumping lemma (4)

We want to show that a language is not regular, we use a proof by contradiction. Let us first remark that  $L$  is necessarily infinite, since all finite languages are regular.

- Assume  $L$  is regular
- Therefore, there exists  $p \geq 1$  (the pumping size) for this language
- There exists a word  $w = xyz$  in the language (with  $y \neq \epsilon$  and  $|xy| \leq p$ )
- Show that, **for all**  $n$ ,  $xy^n z \notin L$
- Contradiction ( $xy^n z$  should be in  $L$ ) therefore  $L$  is not regular

We cannot choose  $p$  nor  $n$  but we can choose  $w$  but we cannot choose how to split it into  $w = xyz$ .



# Pumping lemma (example)

Show that:  $L = \{a^n b^n \mid n \in \mathbb{N}\}$  is not regular. Assume it is.

- there exists  $p \geq 1$  (which we don't know) for which the pumping lemma holds
- we choose  $w = a^p b^p$ . It is a word larger than  $p$  (it has size  $2p$ ).
- choose any  $x, y, z$  such that  $\underbrace{a^i}_x \underbrace{a^j}_y \underbrace{a^{p-i-j} b^p}_z$ , that verifies  $|xy| \leq p$  and  $j \geq 1$ .
- le mot  $xy^0z = a^i a^{p-i-j} 2pb^p = a^{p-j} b^p$  is not in  $L$  (since  $p \geq 1$ , the word has strictly more  $bs$  than  $as$ ). But it should be since all the conditions of the pumping lemma hold. Contradiction,  $L$  is not regular.

- 1 Introduction
- 2 Alphabets, words and word operations
- 3 Languages and their operations
- 4 Regular languages
- 5 Deterministic automata
- 6 Non-deterministic automata
- 7 Minimization
- 8 Advanced concepts
- 9 Conclusions**

We have recapped a lot of things! What should we remember?

- in time, everything!

⇒ *all* these properties translate to trees and tree automata

- For all other formalisms (Grammar, Pushdown automata, ...) some of these properties hold, others don't. When they do hold, often the proofs are similar.