

Informatique Fondamentale IMA S8

Cours 1 - Intro + schedule + finite state machines

Laure Gonnord

<http://laure.gonnord.org/pro/teaching/>
Laure.Gonnord@polytech-lille.fr

Université Lille 1 - Polytech Lille

March/April 2011



Université
Lille1
Sciences et Technologies



POLYTECH[®]
LILLE

Until now

During S5,S6,S7, IMA students have learnt :

- (programming stuff) C programming and compiling,
 - (conception stuff) algorithm design and encoding,
 - (microelec) circuits and embedded systems hardware,
 - (auto) design flows of industrial processes
- ▶ Solved (computation) *problems* by **ad-hoc** solutions.

Until now 2/2

But :

- No general scheme to design algorithms.
 - (manual) evaluation of the *cost* of our programs.
 - **worse** no assurance of correctness.
 - **even worse** it there always a solution ?
- ▶ All these problems will be addressed in this course

Schedule

- Finite state machines (regular automata), regular languages. Notion of non determinism. Link with circuits.
- I/O automata, stack automata and grammars. Link to “simple” languages.
- Counter automata, Turing machines and undecidable problems. Link to “classical” programs.
- Graphs and classical problems/algorithms on graphs.
- Compiler Construction : front end + classical static analysis.
- Compiler Construction : code generation + classical dynamic analysis.

I - Regular languages and automata

- 1 Finite state machines
- 2 Regular Languages
- 3 The notion of non determinism
- 4 Classical Algorithms
- 5 Expressivity of regular languages
- 6 Link to other models

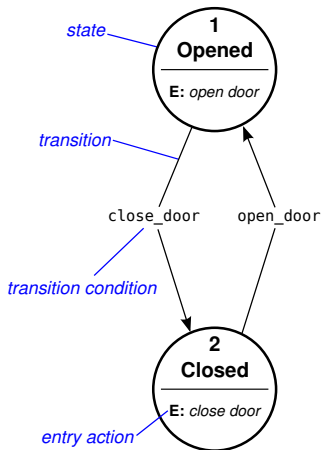
What for ?

We want to model :

- the behaviour of systems with behavioural **modes**.
 - the behaviour of (Boolean) circuits.
 - sets of words.
- ▶ A finite representation.

Example

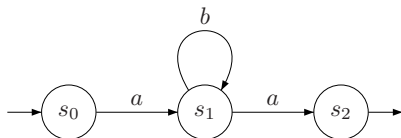
Opening/Closing door - Source Wikipedia.



General definition

Finite state machine (FSM) or **regular automata**

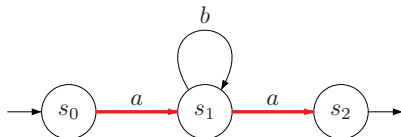
- States are labeled, and there are finitely many ($s \in Q$)
 - Initial state(s) ($i \in I$) and accepting (terminating/finishing) states ($t \in F$)
 - Transitions are finitely many.
 - Transitions are labeled with **letters** ($a \in A$).
- ▶ The transition function is $\delta : Q \times A \rightarrow Q$.



Accepted language 1/2

Accepted word

A **word** w on the alphabet A is **accepted** by the automaton iff there exists a **finite** path from an initial state to an accepting state which is labeled by w .



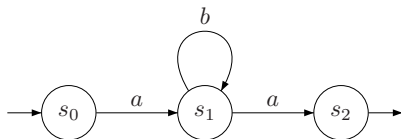
► $w = aa$ is accepted/recognised.

Accepted language 2/2

Accepted language

The **accepted language** of a given automaton \mathcal{A} is the set of all accepted words and is denoted by $\mathcal{L}(\mathcal{A})$.

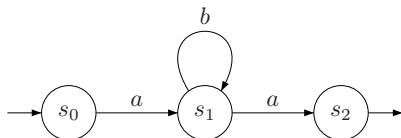
► **Remark** : it can be **infinite**.



► $\mathcal{L}(\mathcal{A}) = \{ab^k a, k \in \mathbb{N}\}$.

Data Structure for implementation

Problem : how to **encode** an automata ?



The transition function can be (for instance) encoded as a **transition table** :

	s_0	s_1	s_2
s_0	--	a	--
s_1	--	b	a
s_2	--	--	--

- 1 Finite state machines
- 2 Regular Languages
- 3 The notion of non determinism
- 4 Classical Algorithms
- 5 Expressivity of regular languages
- 6 Link to other models

Goal

Problem : how to describe languages easily in a textual “linear” way ?

- ▶ use **regular expressions**.

Regular expressions

Regular expression (recursive def)

A **regular expression** e on the alphabet A is defined by induction. It can be of any of the following kinds :

- the empty word ε
- a letter $a \in A$
- a choice between an expression e_1 and another expression e_2 : $e_1 + e_2$
- two successive expressions : $e_1 \cdot e_2$
- 0,1 or more successive occurrences of e_1 : e_1^* .

Example with $A = \{a, b, c, d\}$: $e = a \cdot (b + c \cdot d)^*$

Regular expression vs word

A regular expression “encodes” the form of a word. For instance :

$$i \cdot m \cdot a \cdot (3 + 4 + 5)$$

(on the alphabet $\{i, m, a, 3, 4, 5\}$) describes all words beginning by the prefix “ima” and finishing by one of the numbers 3, 4 or 5.

- ▶ A regular expression describes a **language**

Regular language

Regular language

Given a regular expression e , $\mathcal{L}(e)$ denotes the set of words (the **language**) that are described by the regular expression e .

Example with $A = \{0, \dots, 9\}$:

$e = (1 + 2 + \dots + 9) \cdot (0 + 1 + 2 + \dots + 9)^*$. What is $\mathcal{L}(e)$?

Linux world

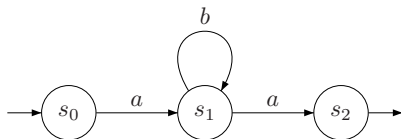
Some commands use (extended) regular expressions (regexp) :

- `ls *.pdf` lists all pdfs of the current directory
- `grep ta*.* *.c` find all lines in .c files that contains words that begin with t + some a's.
- `sed 's ta*.* /toto/g'` file replace all occurrences...

Relationship between automata and languages - 1/3

First, some experiments.

Given the following automata \mathcal{A} , are you able to give a regular expression e such that $\mathcal{L}(e) = \mathcal{L}(\mathcal{A})$?



► $e = ?$

Relationship between automata and languages - 2/3

And the converse :

Given the following regular expression $e = b^* \cdot (c + a)^*$, are you able to give an automaton \mathcal{A} such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(e)$?

Relationship between automata and languages - 3/3

General result - Kleene Theorem

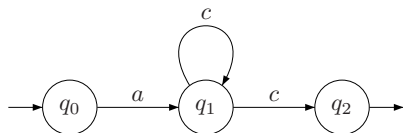
The regular languages are exactly the languages that are described by finite automata.

- ▶ What we've done before is **always** possible.

- 1 Finite state machines
- 2 Regular Languages
- 3 The notion of non determinism
- 4 Classical Algorithms
- 5 Expressivity of regular languages
- 6 Link to other models

Goal

Sometimes some info lacks to make a choice between two transitions :



► From state q_1 , there is a **non deterministic choice** while reading c : either go to state q_2 or stay in q_1 .

Definition

Non deterministic FSM

- A deterministic automaton is $\mathcal{A} = \langle A, Q, I, F, \delta \rangle$ with

$$\delta : A \times Q \rightarrow Q$$

- A **non deterministic** automata is the same with :

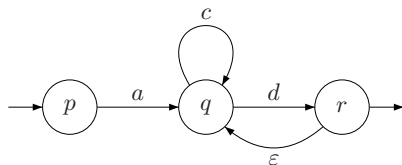
$$\delta : A \times Q \rightarrow P(Q)$$

- A **non deterministic** automata with ε -transitions is the same with

$$\delta : (A \cup \{\varepsilon\}) \times A \rightarrow P(Q)$$

Example

A non deterministic automata with ε -transitions :



► Then $\mathcal{L}(\mathcal{A}) = a \cdot (c^* \cdot d)^*$.

- 1 Finite state machines
- 2 Regular Languages
- 3 The notion of non determinism
- 4 Classical Algorithms**
- 5 Expressivity of regular languages
- 6 Link to other models

Find the associated language

Goal : Given \mathcal{A} an automaton, find the associated language.

▶ Exercises !

Construction of automaton from a regular expression

Goal : Given a regular expression, construct a regular automaton that “**recognises**” it.

▶ Exercises !

Determinisation

Goal : transform a non deterministic automaton into a deterministic one.

▶ Exercises !

Other Algorithms

In the literature you will easily find :

- algorithms to eliminate ε transitions without determinising (ε closure) ;
- algorithms to minimise automata (the number of states) ;
- algorithms to use automata to find words in a text ;
- algorithms to test language inclusion (if they are regular)
- ...

- 1 Finite state machines
- 2 Regular Languages
- 3 The notion of non determinism
- 4 Classical Algorithms
- 5 Expressivity of regular languages**
- 6 Link to other models

Non regular languages

Important result

There exists some **non-regular** languages.

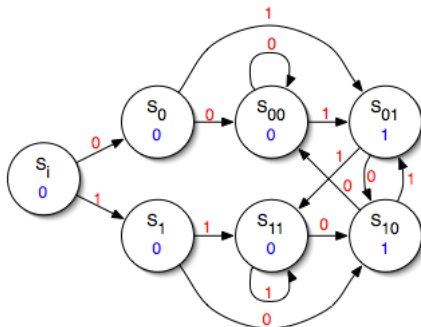
Examples of non-regular languages :

- $\{a^n b^n, n \in \mathbb{N}\}$.
 - palindromes on a non-singleton alphabet.
 - $\{a^p, p \text{ prime}\}$
- ▶ There exists a quite systematic way to prove that a given language is not regular (Pumping Lemma).

- 1 Finite state machines
- 2 Regular Languages
- 3 The notion of non determinism
- 4 Classical Algorithms
- 5 Expressivity of regular languages
- 6 Link to other models

Moore and Mealy Machines - 1

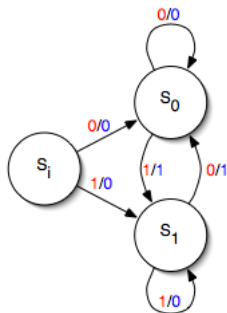
Moore : I/O machine whose output values are determined solely by the current state :



source Wikipedia

Moore and Mealy Machines - 2

Mealy : output values are determined both by the current state and the value of the input.



source Wikipedia

Hardware Implementation of FSMs

It requires :

- a register to store state variables
- a block of combinational logic for the state transition
- (optional) a block of combinatorial logic for the output

Summary

Regular Automata or **Finite State Machines** are :

- Acceptors for regular languages. But some languages are **not regular**.
- Algorithmically efficient.
- Useful to describe (simple) behaviours of systems.
- Closely linked to circuits.