Macchine a responsabilità limitata

Breve Introduzione alla Teoria della Calcolabilità: I Parte - Linguaggi Formali

Roberto Maieli

Università degli Studi "Roma Tre"
maieli@uniroma3.it
http://logica.uniroma3.it/∼maieli/teaching

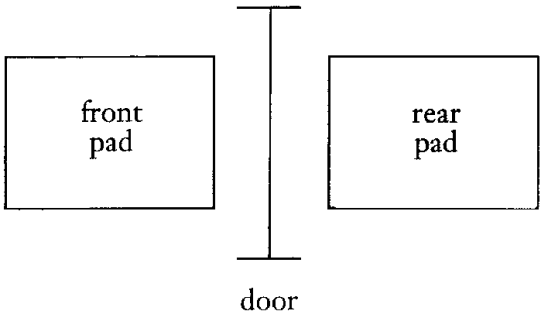
Mini-Corso di II Livello per studenti di Informatica e Filosofia

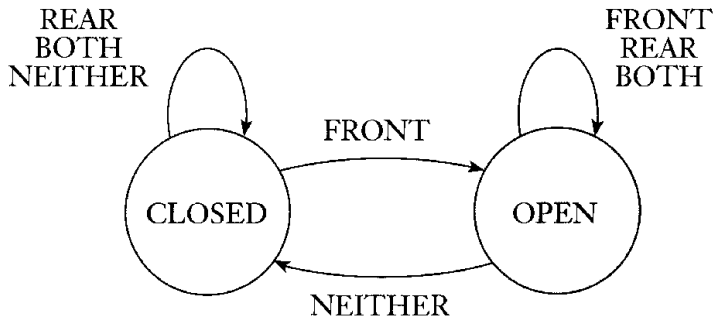
La teoria della computazione nasce con queste domande: cos'è un **algoritmo**? cosa si intende con **modello computazionale**?

- in questo breve corso cercheremo di dare una definizione precisa della nozione intuitiva di algoritmo
- ci approssimeremo a questa nozione costruendo una gerarchia di modelli computazionali (classi di algoritmi) basati su **automi o macchine**.
- come con altri modelli nelle scienze, un modello computazionale può essere accurato per certi versi e per altri no; ecco perchè noi studieremo differenti modelli a seconda delle specifiche richieste.
- mostreremo per ciascuna classe/modello dei "limiti" precisi;
- gli **automi**, a differenza di altri modelli computazionali (λ-*calcolo, funzioni ricorsive*,...):
  - sono modelli **stand-alone**: non presuppongono particolari strumenti matematici (funzioni, riscritture,...)
  - sono **suffcientemente potenti** (equivalenti agli altri modelli),
  - quindi i limiti degli automi valgono anche per gli altri modelli equivalenti.

finite states automata (fsa)

FSA are the **simplest computational model**: they use an extremely limited amounts of memory (states).
What can we do with such a small memory? ... many useful things!
Example the controller for an automatic only-entrance door:





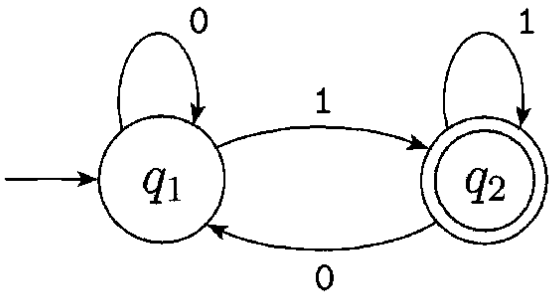
- **states**: OPEN, CLOSED
- **inputs**: FRONT, REAR, BOTH (w.r.t. a person standing on a pad)
- **transition**: it moves from state to state following the recived input

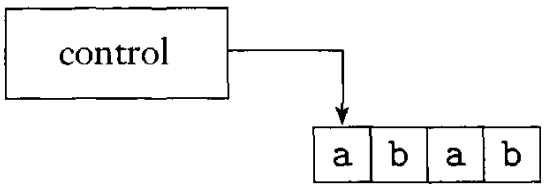
Observe: it makes use of **only one bit of memory**

Definition of FSA

A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta\colon Q \times \Sigma \longrightarrow Q$ is the ***transition function***,[1]
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.[2]





input : it receives the symbols from the input string one by one from left to right,

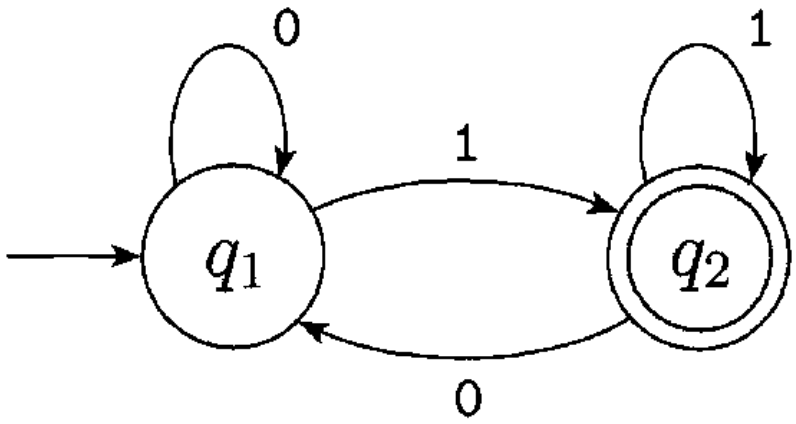
transition : after reading each symbol it moves from one state to an other following (3),

output : when tit reads the last input symbol, it produces the output: **accept** or **reject**

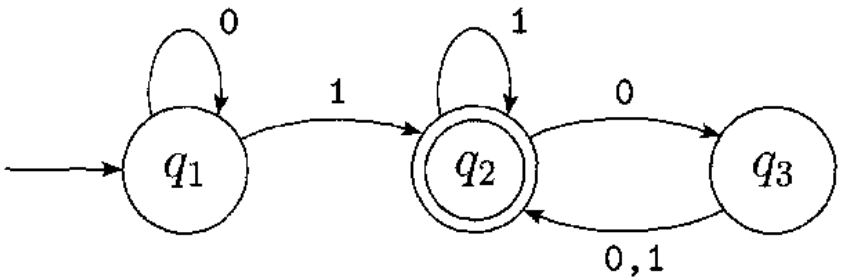
Example of FSA M_0

$M_0 = (\{q_1, q_2\}, \{0, 1\}, \delta, q_1, \{q_2\})$ with the **state diagram** of the transition function δ:

	0	1
q_1	q_1	q_2
q_2	q_1	q_2



Example of FSA M_1



We can describe M_1 formally by writing $M_1 = (Q, \Sigma, \delta, q_1, F)$, where

1. $Q = \{q_1, q_2, q_3\}$,
2. $\Sigma = \{0,1\}$,
3. δ is described as

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

4. q_1 is the start state, and
5. $F = \{q_2\}$.

FSA compute regular languages (RL)

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and let $w = w_1 w_2 \cdots w_n$ be a string where each w_i is a member of the alphabet Σ. Then M ***accepts*** w if a sequence of states $r_0, r_1, \ldots, r_n$ in Q exists with three conditions:

1. $r_0 = q_0$,
2. $\delta(r_i, w_{i+1}) = r_{i+1}$, for $i = 0, \ldots, n-1$, and
3. $r_n \in F$.

Condition 1 says that the machine starts in the start state. Condition 2 says that the machine goes from state to state according to the transition function. Condition 3 says that the machine accepts its input if it ends up in an accept state. We say that M ***recognizes language*** A if $A = \{w | M \text{ accepts } w\}$.

> A language is called a ***regular language*** if some finite automaton recognizes it.

RL: M_0 recognizes $A_0 = \{w \mid w \text{ ends with } 1\}$
M_1 recognizes $A_1 = \{w \mid w \text{ ends with } 1 \text{ or with an even number of } 0\text{s}\}$

FSA are limited: non regular languages

To understand the power of finite automata you must also understand their limitations. In this section we show how to prove that certain languages cannot be recognized by any finite automaton.

Let's take the language $B = \{0^n 1^n | \ n \geq 0\}$. If we attempt to find a DFA that recognizes B, we discover that the machine seems to need to remember how many 0s have been seen so far as it reads the input. Because the number of 0s isn't limited, the machine will have to keep track of an unlimited number of possibilities. But it cannot do so with any finite number of states.

THE PUMPING LEMMA FOR REGULAR LANGUAGES

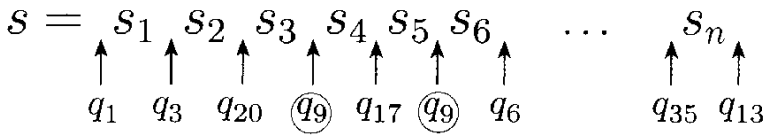
Our technique for proving nonregularity stems from a theorem about regular languages, traditionally called the ***pumping lemma***. This theorem states that all regular languages have a special property. If we can show that a language does not have this property, we are guaranteed that it is not regular. The property states that all strings in the language can be "pumped" if they are at least as long as a certain special value, called the ***pumping length***. That means each such string contains a section that can be repeated any number of times with the resulting string remaining in the language.

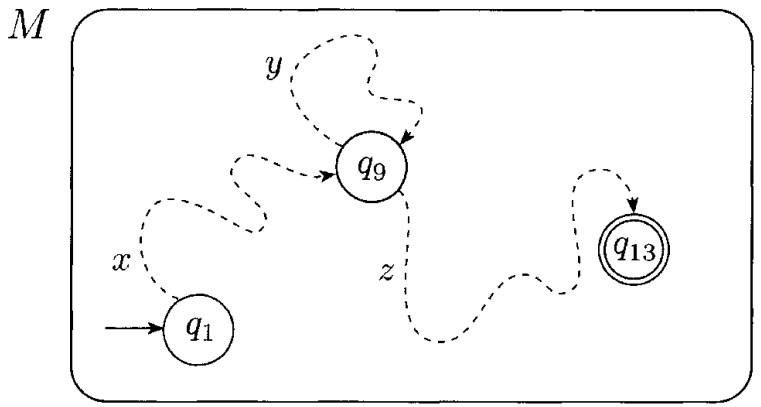
Pumping Theorem for FSA

Assume $M = (Q, \Sigma, \delta, q_1, F)$ be a FSA that recognizes the regular language A and let p be the number or states in Q. If s is a string of A of lenght at least p (i.e. $|s| \geq p$), then s may be divided into three pieces $s = xyz$ such that:

1. for each $i \geq 0, xy^i z \in A$ (y^i means i-copies of y with $y^0 = \epsilon$)
2. $|y| > 0$.

Proof Idea: by hypothesis $|s| = n \geq p$, then the sequence of states that M goes through when computing s is $(n+1) > p$, then this sequence must contain a repeated states (like q_9 below):





the language $B = \{0^n 1^n \mid n \geq 0\}$ is not regular

Assume to the contrary that B is a regular language recognized by M, and choose $s \in B$ s.t. $s \geq p$ (where p is the number of states in M), then by the pumping theorem we can split s into xyz such that for any $i \geq n$ the string $xy^i z \in B$; we consider three cases for $i = 2$:

1. y consits only of 0s, then $xyyz \notin B$ since it has more 0s than 1s
2. y consits only of 1s, then $xyyz \notin B$ since it has more 1s than 0s
3. y consits of both 0s and 1s, then $xyyz \notin B$ since it may have the same number of 0s and 1s but they will not respect the order among 0s and 1s.

some other non-regular languages:
$C = \{w \mid w \text{ has an equal number of 0s and 1s}\}$
$C = \{w.w \mid w \in \{0,1\}\}$.

inductive definition of regular languages

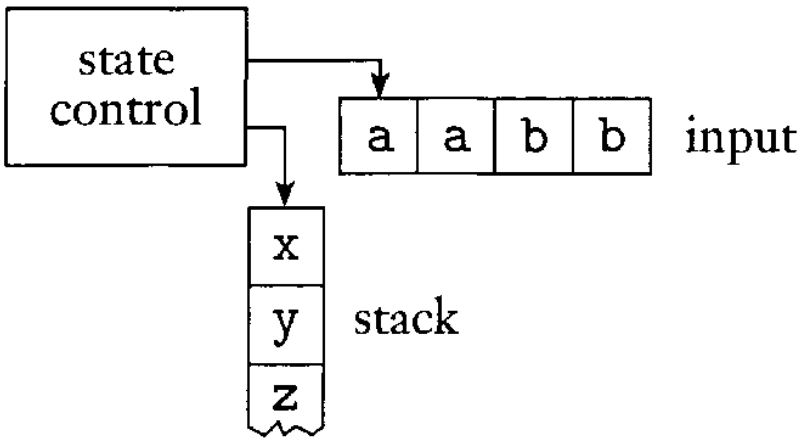
The **pumping theorem** gives a **geometrical characterization** of the class of regular languages against the **inductive one** below:

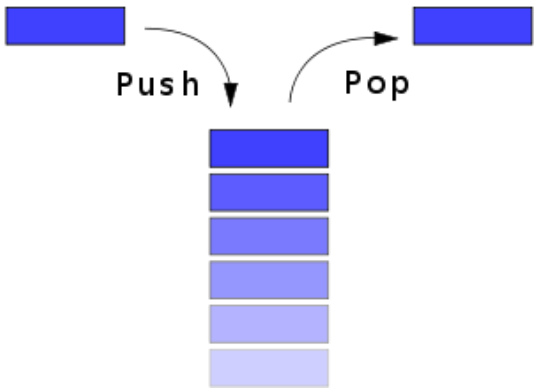
1. $\{a\}$ for any $a \in \Sigma$ (an alfabet) is regular,
2. $\{\epsilon\}$ is regular (ϵ is the empty string),
3. $\emptyset$ is regular (the empty language),
4. $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$ is regular, if A and B are so,
5. $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$ is regular, if A and B are so,
6. $A^* = \{x_1, ..., x_n \mid n \geq 0 \text{ and } x_i \in A\}$ is regular, if A is so.

pushdown automata (PDA)

A PDA is a FSA with an (infinite) **stack** of memory, such that it can:

- **read** from the **input**,
- **write** (*push*) and **read** (*pop*) symbols on the top of the **stack**,
- the stack is "**last in, first out**" storage devise:
  if certain information is written on the stack and additional information is written afterwards, then the earlier information becomes inaccessible until the later information is removed.

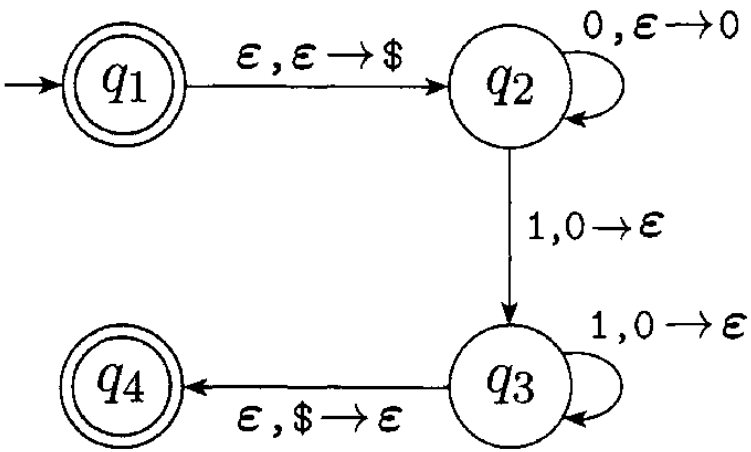




the PDA that recognizes $B = \{0^n 1^n \mid n \geq 0\}$

Let M_1 be the 6-upla $(Q, \Sigma, \Gamma, \delta, q_1, F)$ where:

1. $Q = \{q_1, q_2, q_3, q_4\}$,
2. $\Sigma = \{0, 1\}$,
3. $\Gamma = \{0, \$\}$ ($\$$ initially denotes the *empty stack*)
4. q_1 is the initial state
5. $F = \{q_1, q_4\}$ are the final accepting states,
6. δ is the transition function given by the state diagram below:

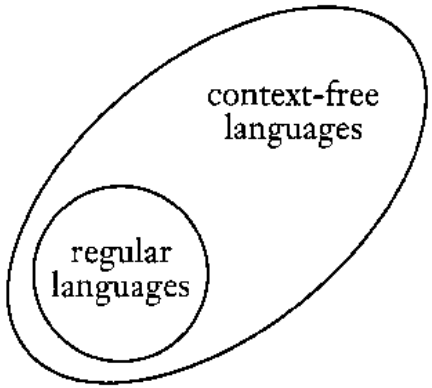


- if "$a, b \to c$" then PDA reads "a from the input and replace "b" from the top of the stack with "c"
- if $a = \epsilon$", it takes transition without reading any input symbol
- if $b = \epsilon$", it takes transition without popping/pushing on the stack
- if $c = \epsilon$", it takes transition without writing on the stack

context-free languages (CFL)

.

- a language recognized by a PDA is called **context-free**



- **question**: is there a "geometrical" characterization of such CFLs?
- **anwer**: the **pumping theorem for PDA** states that every CFL has a special value called **pumping length** s.t. all longer strings in the language can be pumped to other strings (still in the language);
- **question**: is $D = \{w.w \mid w \in \{0,1\}^*\}$ a context-free language?

Pumping Theorem for PDA

If B is a context-free language, then there exists a number p (the **pumping length**) s.t., if s is a string of B of lenght at least p (i.e. $|s| \geq p$), then s may be divided into **five pieces** $s = uvxyz$ such that:

1. for each $i \geq 0, uv^i xy^i z \in B$,
2. $|vy| > 0$

Proof Idea: via the **Chomsky's Lemma**:

any context-free language is generated by a
CONTEXT-FREE GRAMMAR

context-free grammar (Chomsky)

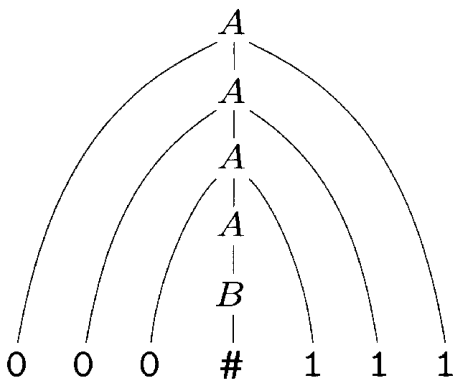
A CFG G_1 consists of :

1. collection of **substitution (or production) rules**: a rule comprises a symbol and a string separated by an arrow "$\rightarrow$"

$$A \rightarrow_1 0A1 \qquad A \rightarrow_2 B \qquad B \rightarrow_3 \sharp$$

2. a symbol A or B is called a **variable**; A is a **start variable**;
3. a **string** consists of variables and **terminal symbols** $0, 1, \#$.
4. the **derivation** of a string $000\#111$ is a **sequence of substitutions**

$$A \Rightarrow_1 0A1 \Rightarrow_1 00A11 \Rightarrow_1 000A111 \Rightarrow_2 000B111 \Rightarrow_3 000\#111$$

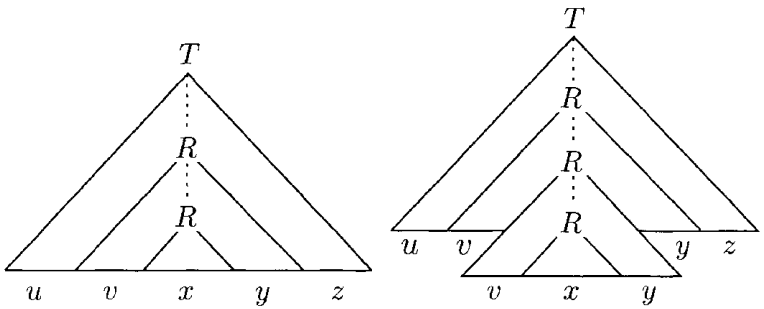


the **parse tree** for the string $000\#111$ in G_1

G_1 **generates** the language $B = \{0^n 1^n \mid n \geq 0\}$

proof idea of the pumping lemma for PDA

1. if B is a CFL, by Chomsky's lem., there is a CFG G that generates B
2. let's choose p big enough that the parser tree of s (with $|s| \geq p$) is so tall that it must contain some long path from the start variable at the root to one of the terminal symbol at a leaf with a repetition of some variable R
3. we replace the subtree rooted at the lower occurrence of R with the subtree rooted at the higher occurrence of R and get a tree of a string $s' = uv^2xy^2z$ still in B.



[1] if $p = b^{|V|+2}$, where b is the maximum number of symbols on the right hand side of a rule of G and $|V|$ is the number of variables V in G, then $h_T \geq |V| + 2$, so there is a R repetead in T.

a non-context free language

question: is $D = \{w.w \mid w \in \{0,1\}^*\}$ a context-free language?
answer: NO! (e.g., strings like $0^p1^p.0^p1^p$ cannot pumped)

CFGs can describe certain features that have a **recursive structure**, which makes them useful in a variety of applications:

- LINGUISTICS: they were first used in the study of human languages: a first attempt of understanding the relationship of terms like **noun, verb, preposition** and their respective **phrases** leads to a natural recursion because **noun phrases** may appear inside **verb phrases** and viceversa

$$[(\texttt{the boy})^{\mathbf{NP}}\ ((\texttt{sees})^{\mathbf{VP}}\ (\texttt{a flower})^{\mathbf{NP}})^{\mathbf{VP}}]^{\mathbf{S}}$$

- COMPUTER SCIENCE: most compilers and interprets of programming languages (C, Java, ...) contain a **parser** that extracts the *meaning of a program* before generating the compiled code or performing the interepreted execution.