Linguaggi formali

Let's start from the beginning

- A program is written in a programming language
- Every programming language (as every language in general) needs to obey its own rules
- We need to formally define languages…

Reference books

Intoduction to Automata Theory, Languages, And Computation.
Hopcroft, Motwani, Ullman

Fondamenti dell'Informatica. Linguaggi formali, calcolabilita' e complessita'.
Dovier, Giacobazzi
Bollati Boringhieri

Strings

- An alphabet is a finite set of symbols
- Examples

Σ_1 = {a, b, c, d, …, z}: the set of letters in Italian

Σ_2 = {0, 1}: the set of binary digits

Σ_3 = {(,)}: the set of open and closed brackets

A string over alphabet Σ is a finite sequence of symbols in Σ.

- Examples

abfbz is a string over Σ1 = {a, b, c, d, …, z}

11011 is a string over Σ2 = {0, 1}

))()(() is a string over Σ3 = {(,)}

The empty string is a string having no symbol, denoted by ε.

Strings

- The length of a string x is the number of symbols contained in the string x, denoted by |x|.

- Examples

  |abfbz|=5

  |110010|=6

  |))()(()|=7

  |ε|=0

Strings

- The concatenation of two strings x and y is a string xy,
  i.e., x is followed by y.
  it is an associative operation that admits the neutral element ε

- s is a substring of x if there exist strings y and z
  such that x = ysz.

Example:

the prefixes of abc are : ε, a, ab, abc

- In particular,
  when x = sz (substring with y=ε), s is called a prefix of x;
  when x = ys (substring with z=ε), s is called a suffix of x;

ε is a prefix and a suffix of ε and of all strings

Power of an alphabet

- We indicate the set of all strings over Σ of a given length

Σ^n denotes the strings of length n whose symbols are in Σ

If $\Sigma=\{0,1\}$

$\Sigma^0 = \{\varepsilon\}$

$\Sigma^1 = \Sigma = \{0,1\}$

$\Sigma^2 = \{00,01,11,10\}$

$\Sigma^3 = \{000,001,010,011, 100,101,110,111\}$

$$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \Sigma^4 \cup \ldots = \bigcup_{i>0} \Sigma^i \qquad \Sigma^* = \{\epsilon\} \cup \Sigma^+$$

$\Sigma^+ = \{0,1,00,01,11,10,000,001,010,011, 100,101,110,111....\}$

Languages

A language is a set of strings over an alphabet:

$L \subseteq \Sigma^*$ is a language over Σ

Examples

L_1 = The set of all strings over Σ_1 that contain the substring "fool"

L_2 = The set of all strings over Σ_2 that are divisible by 7

= {7, 14, 21, …}

L_3 = The set of all strings over Σ_3 where every (is followed by 2 occurrences of)

= {(),)),)()), …}

Other examples of Languages

L_4 = The set of binary numbers whose value is prime

{ 10,11,101,111,1011,1101,...}

L_5 = The set of legal English words over the English alphabet

L_6 = The set of legal C programs over the strings of characters

Languages

- The following are operations on sets and hence also on languages.

  Union: $A \cup B$

  Intersection: $A \cap B$

  Difference: $A \setminus B$ ($A - B$ when $B \subseteq A$)

  Complement: $A = \Sigma^* - A$ where Σ^* is the set of all strings on alphabet Σ.

  Concatenation: $AB = \{ab \mid a \in A, b \in B\}$

Example: $\{0, 1\}\{1, 2\} = \{01, 02, 11, 12\}$.

Kleene Clousure

Kleene closure: $$A^* = \bigcup_{i=0}^{\infty} A^i$$

- Notation: $$A^+ = \bigcup_{i=1}^{\infty} A^i$$

More example of Languages

Examples:

- The set of strings with n 1's followed by n 0's
  {ε, 01, 0011, 000111, . . .}

- The set of strings with an equal number of 0's and 1's
  {ε, 01, 10, 0011, 0101, 1001, . . .}

- The empty language ∅

- The language {ε} consisting of the empty string only

Remember $\varnothing \neq \{\varepsilon\}$

Problems

- Does the string w belong to the language L?

  Example: $11101 \in L_4$?

We want to define a procedure to decide it!

We can try to generate all words....

We can try to recognise when a word belongs to a Language

Generating a language: Grammars

Starting from a particular initial symbol, using the rewriting rules of the productions,

we generate the set of strings belonging to the language

Grammars I

We define a Grammar $G=(\Sigma, N, S, P)$ where :

- Σ is the alphabet, a set of symbols (called terminals)
- N is the set of nonterminals
- $S \in N$ is the starting symbol
- P is the set of productions, each of the form

$$U \rightarrow V$$

where $U \in (\Sigma \cup N)^{+}$ and $V \in (\Sigma \cup N)^{*}$.

Grammars II

$G = (\Sigma, N, S, P)$

A string $w \in \Sigma$ is generated by G if there is a derivation of w using P, starting from the starting symbol S.

$G = (\{a\}, \{S\}, S, P)$

$S \rightarrow \varepsilon$

$S \rightarrow a$

$S \rightarrow aS$

A language generated by grammar G is denoted $L(G)$ and it is the set of strings derived using G.

Grammar Example

We want to describe L1 the language of strings with an even number of 1's

L1 can be generated by a grammar ({0,1},{S,T},S,P) with P equal to

$S \rightarrow \varepsilon$

$S \rightarrow 0S$

$S \rightarrow 1T$

$T \rightarrow 0T$

$T \rightarrow 1S$

A string belongs to L1 iff it can be generated by the grammar

Grammar Example

Does the string 01010 belong to L1?

We need to find a derivation

$S \to \varepsilon \mid 0S \mid 1T$

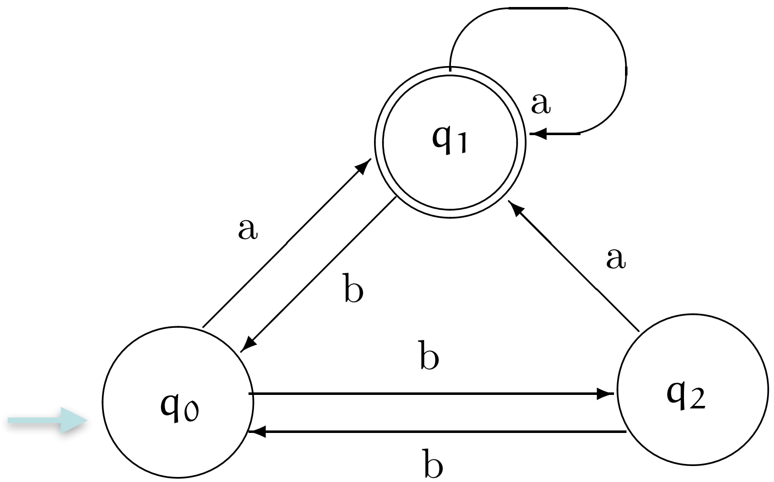
$T \to 0T \mid 1S$

S

Recognising a language: Automata

- A finite state automaton is finite state machine with an input of discrete values.
- The state machine consumes the input and possibly moves to a different state.
- The system may be in a state among a finite set of possible states. Being in a state allows him to keep track of previous history.

input: baab



Back to our Problems

- Does the string w belong to the language L?

  We want to define a procedure to decide it!

- Which is the computational complexity necessary to answer to the previous question ?

It depends on the complexity of the language!!

Grammars and Languages

Restrictions on productions give different types of grammars :

- Regular (type 3)
- Context-free (type 2)
- Context-sensitive (type 1)
- Phrase-structure (type 0)

$$U \rightarrow V$$

where $U \in (\Sigma \cup N)+$ and $V \in (\Sigma \cup N)^*$.

For context-free, e.g., $U \in N$

No restrictions for phrase-structure

A language is of type i iff

there is a grammar of type i which describes it

Complexity of Languages Problems

	Regular Grammar Type 3	Context Free Grammar Type 2	Context Sensitive Grammar Type 1	Unrestricted Grammar Type 0
Is $W \in L(G)$?	P	P	PSPACE	U
Is L(G) empty?	P	P	U	U
Is $L(G1) \equiv L(G2)$?	PSPACE	U	U	U

P: decidable in polynomial time

PSPACE: decidable in polynomial space (at least as hard as NP-complete)

U: undecidable

Regular languages

All the following ways to represent regular languages are equivalent:

- Regular grammars (RG, type 3)
- Deterministic finite automata (DFA)
- Non-deterministic finite automata (NFA)
- Non-deterministic finite automata with ε transitions (ε-NFA)
- Regular expressions (RE)

Regular Grammars

A Right (or, analogously, Left) Regular Grammar is a generative grammar, where

- every production has the form A-> aB| a
- only for the starting symbol S, we can have $S \rightarrow \varepsilon$

every non terminal symbol B is always preceded by a terminal one.

Example

G=({a,b}, {S,B},S,P) where productions P are:

S-> aS|aB

B->bB|b

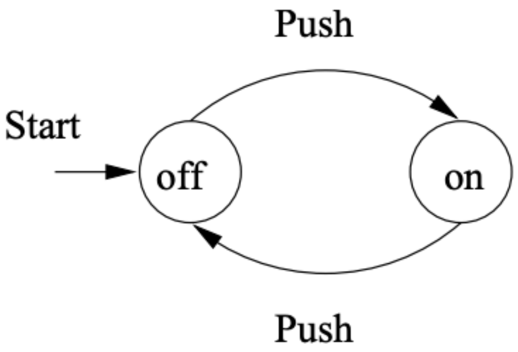
aabb $\in$ L(G)

L(G)={ $a^n b^m$| n,m>0}

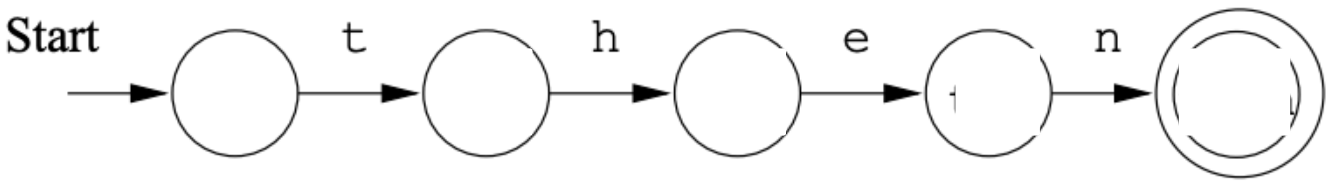
S

Deterministic Finite Automata

The states of a switch:



An automaton recognising the keyword then:



Deterministic Finite Automata

A deterministic finite automaton (DFA) $(Q, \Sigma, \delta, q_o, F)$

Q a finite set of states

Σ a finite set Σ of symbols

$\delta : Q \times \Sigma \rightarrow Q$ a transition function that takes as argument a state and a symbol and returns one state

q_o the starting state

$F \subseteq Q$ the set of final or accepting states

Deterministic Finite Automata

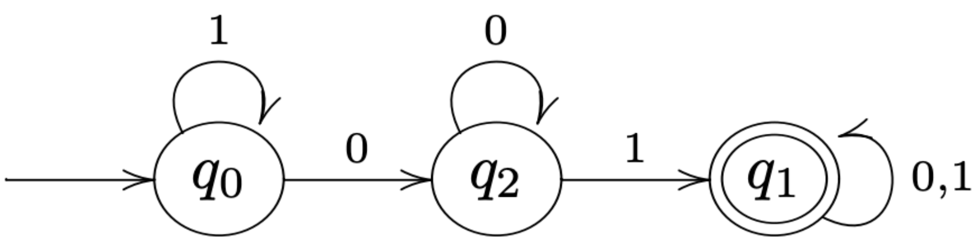
How to represent a DFA? With a transition table

	0	1
$\rightarrow q_0$	q_2	q_0
$*q_1$	q_1	q_1
q_2	q_2	q_1

-> indicates the starting state

* indicates the final states

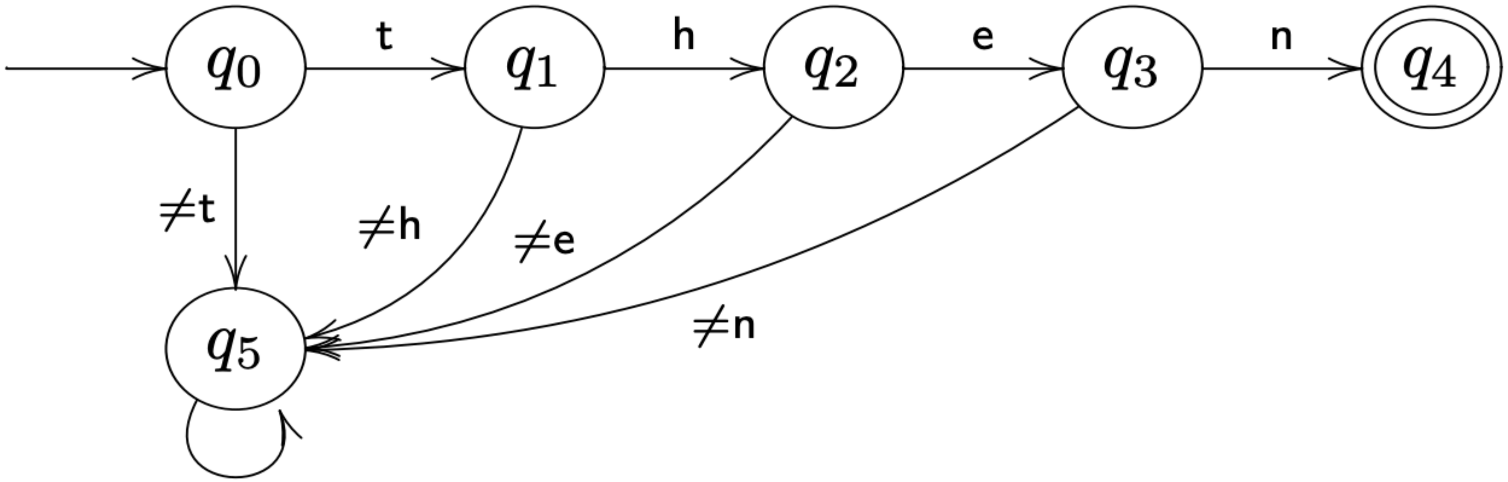
This defines the following transition diagram



Deterministic Finite Automata

When does an automaton accept a word?

It reads a word and accept it if it stops in an accepting state



here Q= $\{q_0, q_1, q_2, q_3, q_4, q_5\}$ F= $\{q_4\}$

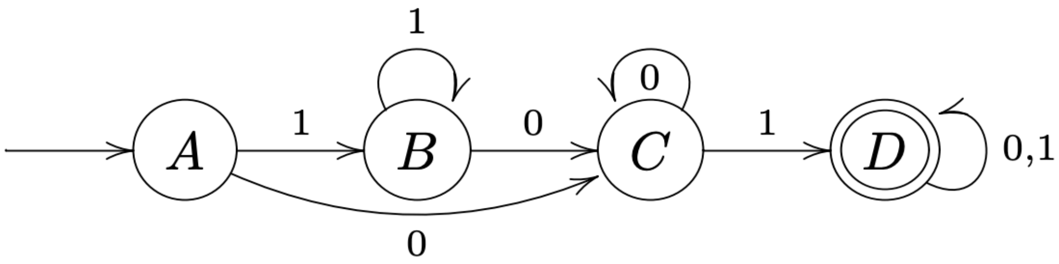
Only the word then is accepted

How DFA processes Strings

We build an automaton that accepts string containing the substring 01

Σ={0,1}
L={x01y| x,y$\in\Sigma^*$}

We get



	0	1
→A	C	B
B	C	B
C	C	D
*D	D	D

Extending the transition function to Strings

We define the transitive closure of δ

$$\hat{\delta} : Q \times \Sigma^* \longrightarrow Q$$

$$\begin{cases} \hat{\delta}(q, \varepsilon) & = & q \\ \hat{\delta}(q, wa) & = & \delta(\hat{\delta}(q, w), a) \end{cases}$$

A string x is accepted by M=(Q, Σ, δ ,qo,F) iff $\widehat{\delta}(q_0, x) \in F$

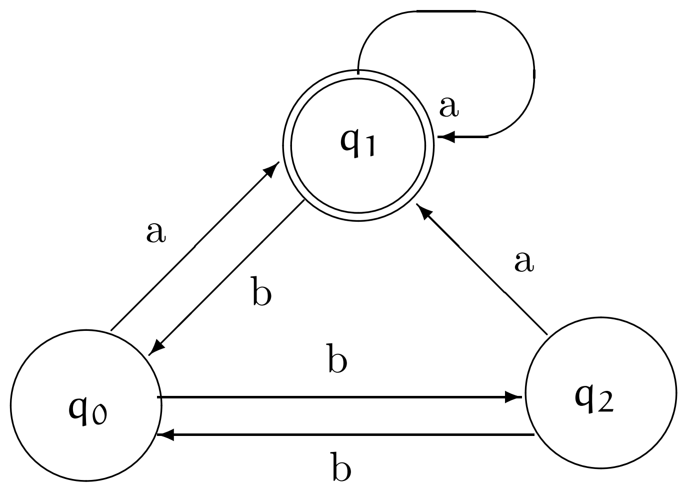
$$L(M) = \{x \in \Sigma^* | \widehat{\delta}(q_0, x) \in F\}$$

Nondeterministic Finite Automata

A nondeterministic finite automaton (NFA) allows more than one transition on the same input symbol.

Formally, a NFA is defined as (Q, Σ, δ ,q_0, F) where the only difference is the transition function

$\delta : Q \times \Sigma \rightarrow \wp(Q)$ a transition function that takes as argument a state and a symbol and returns a set of states



Extending the transition function to Strings

We define the transitive closure of δ

$$\begin{cases} \hat{\delta}(q, \varepsilon) & = & \{q\} \\ \hat{\delta}(q, wa) & = & \bigcup_{p \in \hat{\delta}(q,w)} \delta(p, a) \end{cases}$$

□

A string x is accepted by M=(Q, Σ, δ ,qo,F) iff $\widehat{\delta}(q_0, x) \cap F \neq \emptyset$

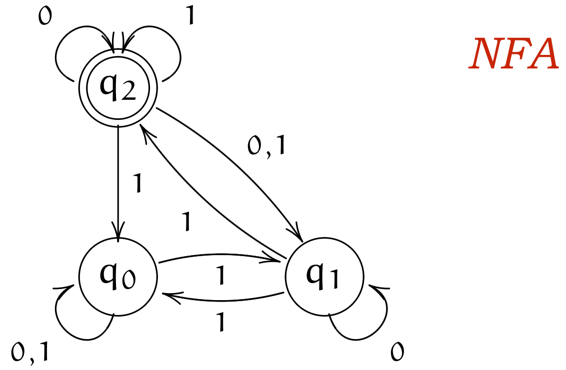
$$L(M) = \{x \in \Sigma^* | \widehat{\delta}(q_0, x) \cap F \neq \emptyset\}$$

- NFAs do not expand the class of language that can be accepted.

Example

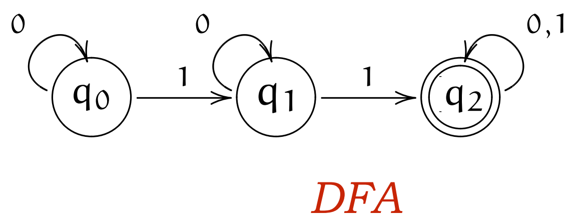
		0	1
→	q_0	$\{q_0\}$	$\{q_0, q_1\}$
	q_1	$\{q_1\}$	$\{q_0, q_2\}$
*	q_2	$\{q_1, q_2\}$	$\{q_0, q_1, q_2\}$

$F = \{q_2\}$



L= $\{x \in \{0,1\}^* \mid$ x contains at least 2 occurrences of 1 }

		0	1
→	q_0	q_0	q_1
	q_1	q_1	q_2
*	q_2	q_2	q_2

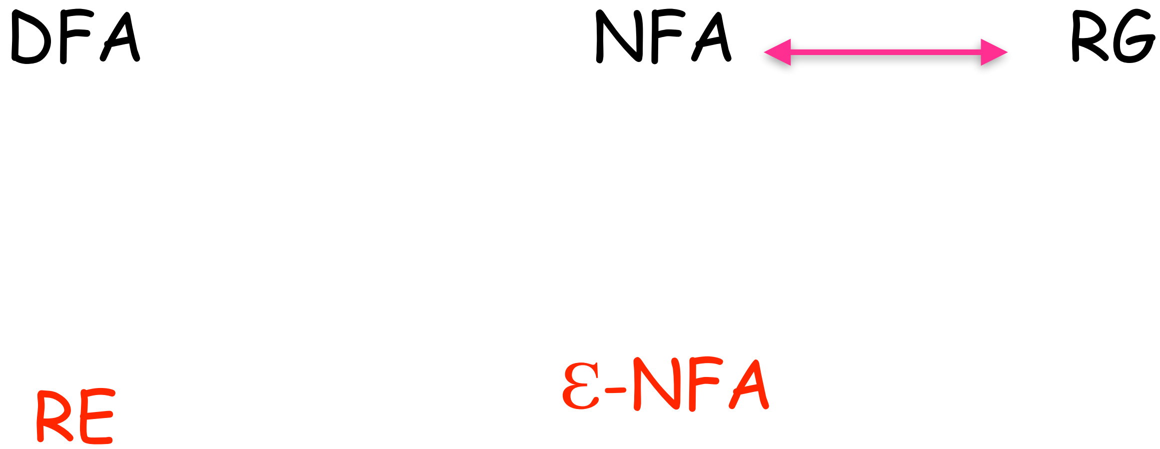


Different characterisation of Regular Languages

There are different ways to characterise a regular language

- Regular grammars
- Deterministic Finite Automata
- Non Deterministic Finite Automata
- Epsilon Non deterministic Finite Automata
- Regular expression

Roadmap: equivalence between NFA and RE



From Regular Grammars to NFA

Theorem 1.

For each right grammar RG (or left grammar LG), there is a non deterministic finite automaton NFA such that L(RG)=L(NFA).

Construction Algorithm

For a given right grammar RG=(Σ, N, S, P) there is a corresponding NFA=(N U{F}, Σ, δ,S, F') where F is a newly added state and if F'= {F}U{S} if S-> ε belongs to P, F'= {F}, otherwise.

The transition function δ is defined by the following rules.

1) For any A->a belonging to P, with a in Σ, set δ(A,a) = F
2) For any A-> aB belonging to P, with a in Σ and B in N, set δ(A,a)=B

Example

G=({a,b}, {S,B},S,P) where productions P are:

S-> aS|aB

B->bB|b $\qquad$ L(G)={ $a^n b^m$ | n,m>0}

From NFA to Regular Grammars

Theorem 2

For each nondet finite automaton NFA, there is one right grammar RG (or left grammar LG) where L(RG)=L(NFA).

For a given finite automata NFA= (Q, Σ, δ , q_0, F), a corresponding right grammar RG=(Σ, Q, q_0', P) can be constructed using the following steps

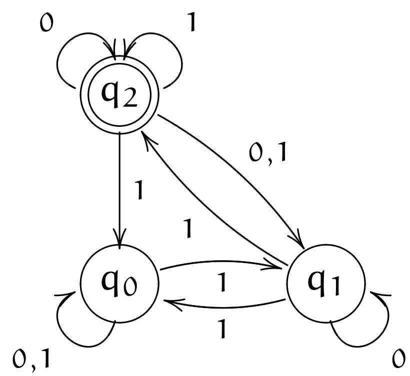
1) for any $\delta(A,a)=B$ add $A \rightarrow aB$ to P,
2) if B belongs to F add also $A \rightarrow a$ to P;

If q_0 belongs to F then add $q \rightarrow q_0 \mid \varepsilon$ to P and $q_0'=q$ else $q_0'=q_0$.

Example

	0	1
→ q_0	$\{q_0\}$	$\{q_0, q_1\}$
q_1	$\{q_1\}$	$\{q_0, q_2\}$
* q_2	$\{q_1, q_2\}$	$\{q_0, q_1, q_2\}$

$F = \{q_2\}$.



NFA

L= $\{x \in \{0,1\}^* \mid$ x contains at least 2 occurrences of 1 }

Exercises

Write the NFA for the following languages

- The set of string over the alphabet {a,b,c} containing at least one a and at least one b
- The set of strings of 0's and 1's whose tenth symbol from the right is 1
- The set of strings of 0's and 1's with at most one pair of consecutive 1's

and derive the corresponding grammars