

Automati a stati finiti

- **Il modello: la definizione formale, esempi.**
- **Le definizioni utili per descrivere e provare proprietà degli automi: configurazioni, relazione “porta a” e relative definizioni di linguaggio accettato.**

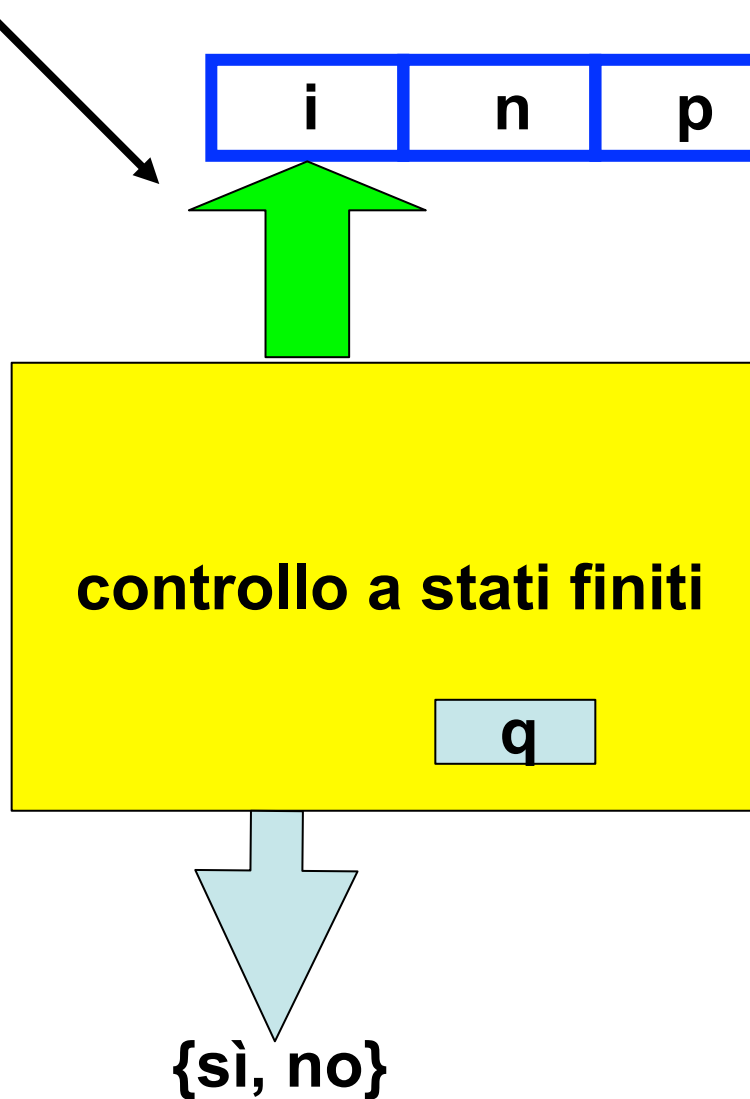
Automati a stati finiti

Hanno applicazioni in

- verifica dei circuiti digitali e dei protocolli,
- compilatori,
- pattern recognition,
- etc.

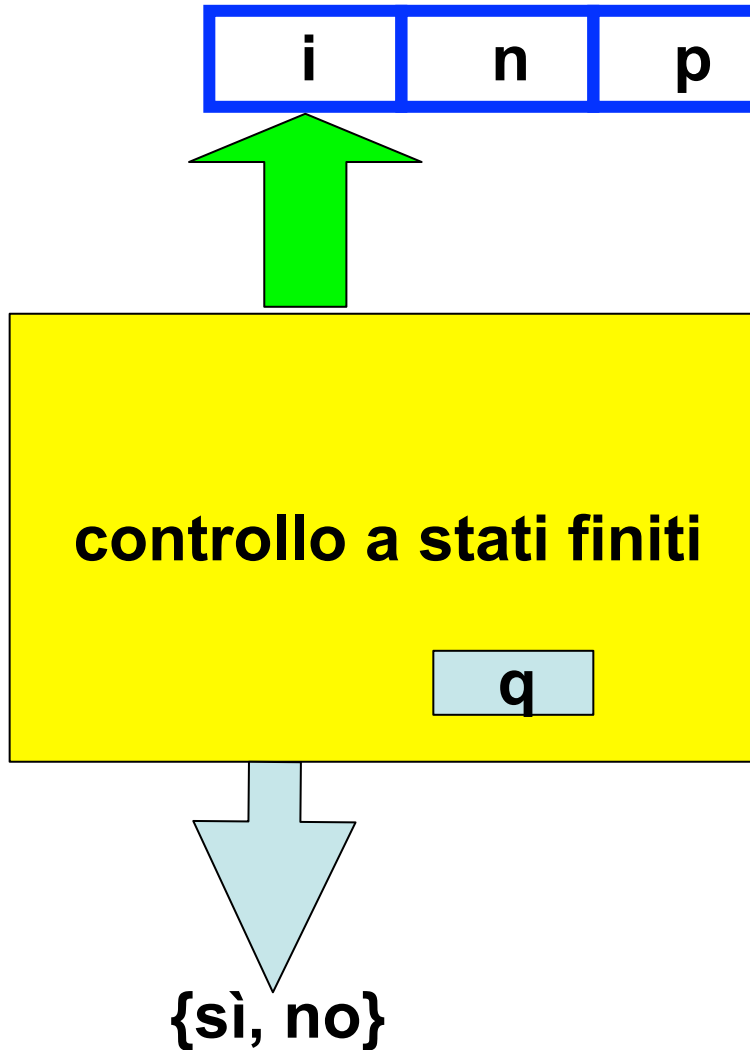
II MODELLO MENTALE

testina di lettura,
può muoversi
solo verso destra



In funzione dello stato in cui si trova e dell'input letto la macchina cambia stato, poi sposta la testina di lettura sulla cella successiva. Termina dopo aver letto l'ultimo simbolo sul nastro input; lo stato raggiunto determina se la parola è accettata o no.

Un esempio di esecuzione



La macchina incorpora un programma con istruzioni del tipo:

nello stato q se leggi i vai nello stato p

nello stato p se leggi n vai nello stato q ...

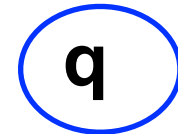
A ogni passo la testina di lettura si sposta di una cella a destra.

La macchina si ferma quando ha letto tutto l'input.

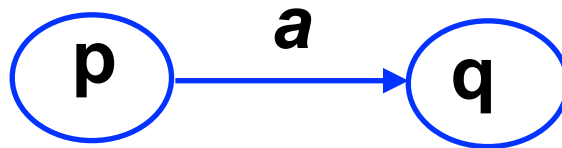
DIAGRAMMA DEGLI STATI

È un grafo diretto in cui

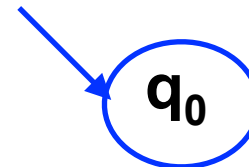
- ogni nodo corrisponde a uno stato



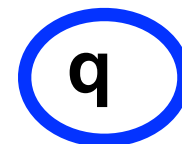
- tra una coppia di nodi corrispondenti agli stati p e q c'è un arco con etichetta a se c'è la regola nello stato p se leggi a vai nello stato q



- lo stato iniziale q_0 è denotato

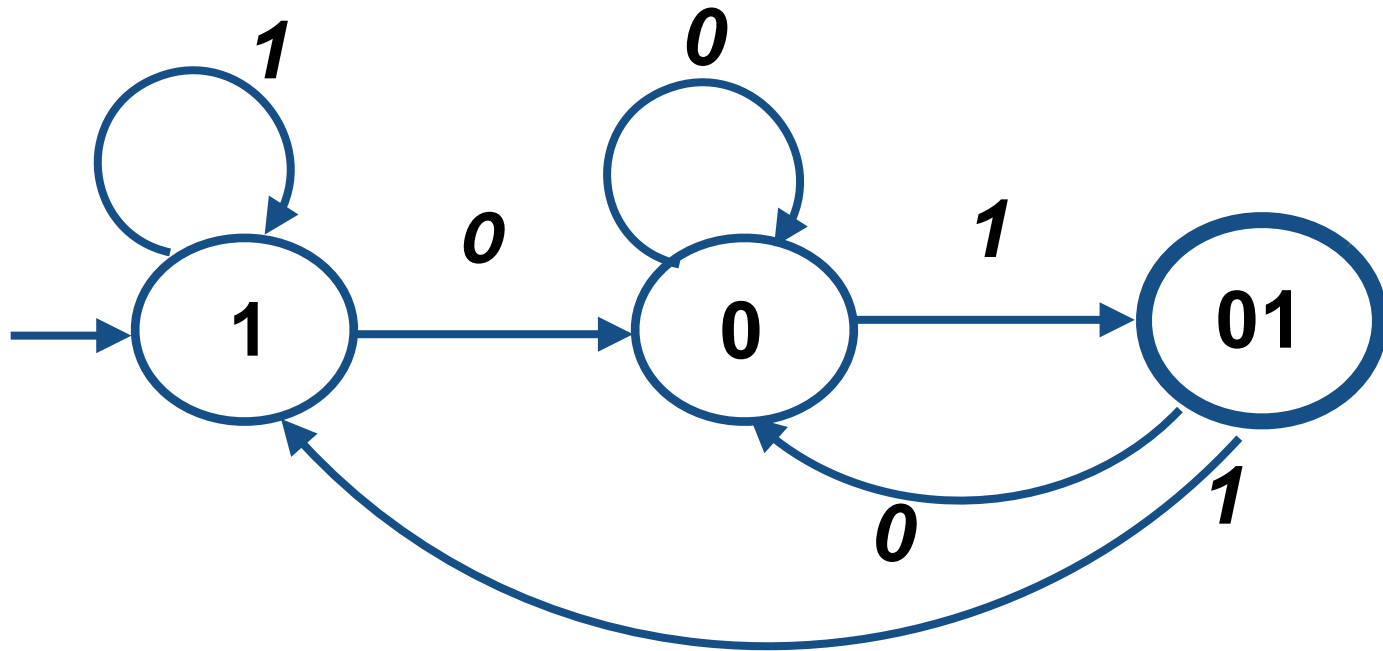


- uno stato finale q è denotato



Esempio 1

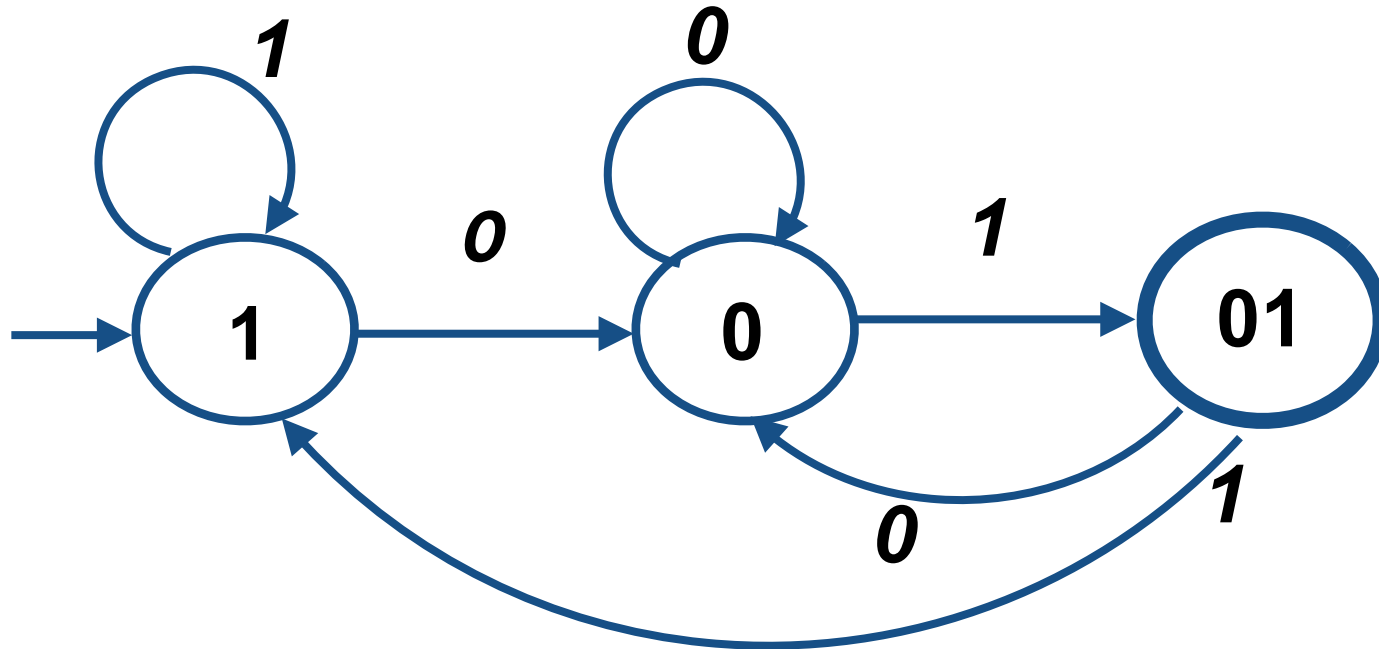
Vogliamo costruire un DFA che accetta tutte le stringhe binarie che terminano con 01



Ogni stringa che determina un cammino sul grafo diretto dallo stato iniziale a quello finale è accettata dall'automa

Esempio 1

Ogni stringa binaria w determina un cammino di lunghezza $|w|+1$, la lunghezza di w più 1



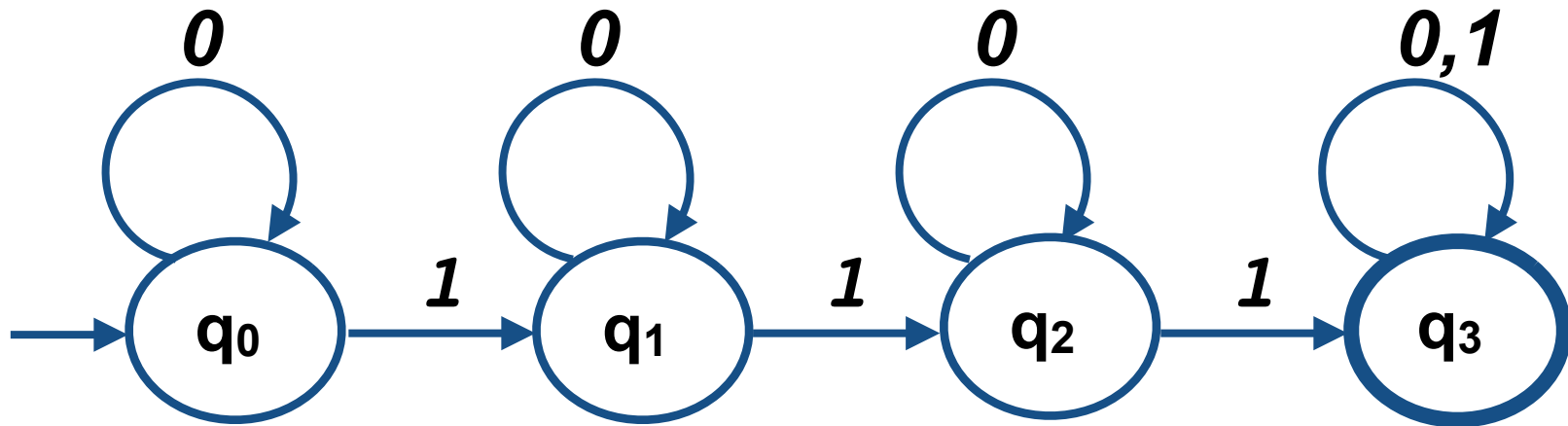
Definizione ricorsiva di lunghezza:

$$|\varepsilon| = 0$$

$$|wa| = |w|+1, \text{ per } a \text{ in } \{0,1\}$$

Esempio 2

Vogliamo costruire un DFA che accetta tutte le stringhe binarie che contengono almeno tre 1



la stringa

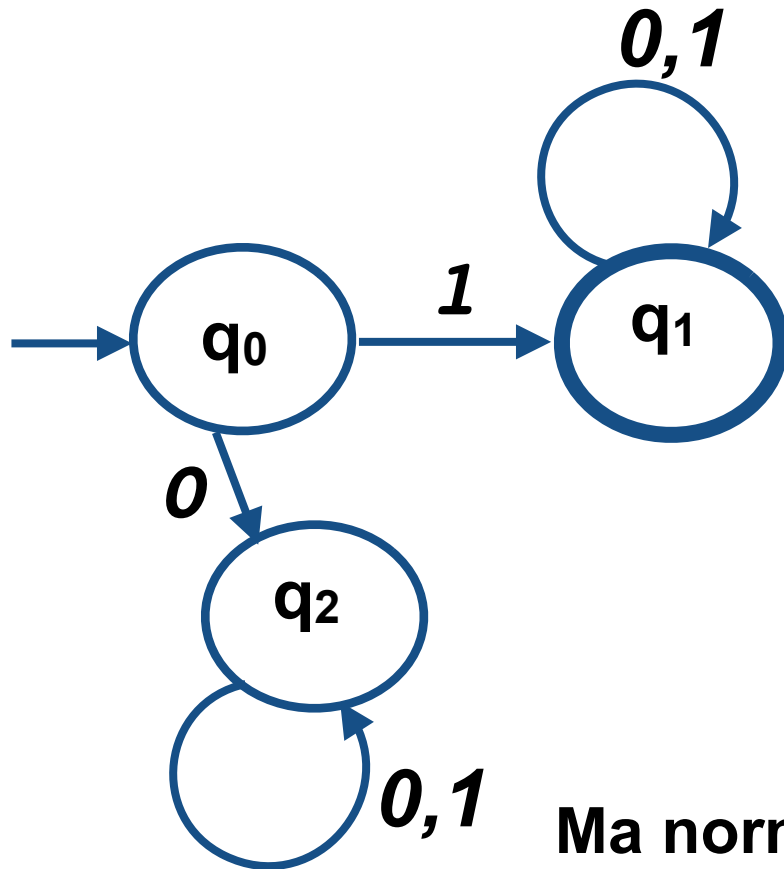
0 0 1 0 1 1 0 0 determina il cammino

q₀ q₀ q₀ q₁ q₁ q₂ q₃ q₃ q₃

Ci sono tanti archi uscenti da ogni nodo quante sono le lettere dell'alfabeto

Esempio 3

Vogliamo costruire un DFA che accetta tutte le stringhe binarie che cominciano per 1



E la parola 010?

Così manteniamo la proprietà che ogni nodo ha tanti archi uscenti quanti sono gli elementi dell'alfabeto e di conseguenza quella per cui ogni parola determina un cammino sul grafo, proprietà comoda nelle costruzioni

Ma normalmente non si disegna questa parte dell'automa

Modello formale

Un **automa a stati finiti deterministico**, in breve DFA (Deterministic Finite Automaton), è una quintupla

$M = (Q, \Sigma, \delta, q_0, F)$ dove

- Q è l'insieme finito degli stati;
- Σ è l'insieme finito dei simboli di input;
- $\delta : Q \times \Sigma \rightarrow Q$ è la funzione (totale) di transizione;
- q_0 è lo stato iniziale;
- $F \subseteq Q$ è l'insieme degli stati finali o di accettazione

CONFIGURAZIONI

Lo stato di una macchina descrive la condizione in cui si trova il controllo a stati finiti, la **configurazione** descrive anche lo stato dei dispositivi input e, se presenti, delle memorie aggiuntive.

Una **configurazione** di un DFA $M = (Q, \Sigma, \delta, q_0, F)$ indica lo stato in cui si trova la macchina e la porzione di input ancora da leggere, è quindi un elemento di $Q \times \Sigma^*$.

Per un input $x \in \Sigma^*$, la **configurazione iniziale** o **di partenza** è (q_0, x) .

Descrizione calcolo via configurazioni

La relazione "**porta a**" (in un passo) \Rightarrow_M è una relazione binaria su $Q \times \Sigma^*$, così definita:

$$(p, ax) \Rightarrow_M (q, x) \text{ se e solo se } \delta(p, a) = q$$

dove $p, q \in Q$, $a \in \Sigma$ e $x \in \Sigma^*$.

Esempio

δ	0	1
q₀	q ₂	q ₁
q₁	q ₁	q ₁
q₂	q ₂	q ₂

$$(q_0, \mathbf{101}) \Rightarrow_M (q_1, \mathbf{01})$$

Tutto il calcolo su **101** è descritto come segue:

$$(q_0, \mathbf{101}) \Rightarrow_M (q_1, \mathbf{01}) \Rightarrow_M (q_1, \mathbf{1}) \Rightarrow_M (q_1, \epsilon)$$

Chiusura riflessiva e transitiva di una relazione binaria

Data una relazione binaria R su X , cioè $R \subseteq X^2$, la chiusura riflessiva e transitiva di R si ottiene semplicemente aggiungendo a R quello che manca perché sia riflessiva e transitiva, quindi:

$$R^* = R \cup$$

$\{(x,x) \mid x \text{ è in } X\}$ (per la **riflessività**) \cup

$\{(x,z) \mid \exists y (x,y) \text{ e } (y,z) \text{ sono in } R\}$ (per la **transitività**)

Esempio: $X = \{a,b,ab,ba\}$, $R = \{(a,ab),(ab,ba)\}$,

$$R^* = R \cup \{(a,a),(b,b),(ab,ab),(ba,ba)\} \cup \{(a,ba)\}$$

Il caso della “porta a”

Nel caso di nostro interesse X è l'insieme delle configurazioni $Q \times \Sigma^*$ e R è la relazione “porta a”, \Rightarrow_M , e la chiusura riflessiva e transitiva consente di evitare di descrivere il calcolo nel dettaglio degli stati intermedi.

La riflessività esprime il fatto che senza leggere un input la macchina non cambia stato:

$$(q, \mathbf{x}) \Rightarrow_{M^*} (q, \mathbf{x}), \text{ per ogni } q \in Q \text{ e } \mathbf{x} \in \Sigma^*$$

la transitività consente di esprimere succintamente due o più passi di calcolo:

$$\text{se } (p, \mathbf{aby}) \Rightarrow_M (q, \mathbf{by}) \text{ e } (q, \mathbf{by}) \Rightarrow_M (r, \mathbf{y}) \text{ allora } (p, \mathbf{aby}) \Rightarrow_{M^*} (r, \mathbf{y}).$$

Esempio

δ	0	1
q₀	q ₂	q ₁
q₁	q ₁	q ₁
q₂	q ₂	q ₂

La computazione su **10111111** è descritta succintamente come segue:

$$(q_0, \mathbf{10111111}) \Rightarrow_M^* (q_1, \mathbf{\epsilon})$$

DEFINIZIONE FORMALE DI ACCETTAZIONE - 1

Una parola $x \in \Sigma^*$ è accettata da un DFA

$M = (Q, \Sigma, \delta, q_0, F)$ se

$$(q_0, x) \Rightarrow_M^* (q, \varepsilon) \text{ e } q \in F.$$

Il **linguaggio** $L(M)$ **accettato** dal DFA M è l'insieme di tutte le parole accettate da M :

$$L(M) = \{x \in \Sigma^* \mid (q_0, x) \Rightarrow_M^* (q, \varepsilon) \text{ e } q \in F\}$$

FUNZIONE DI TRANSIZIONE ESTESA

Per indicare lo stato raggiunto a partire dallo stato iniziale su un input qualsiasi introduciamo **l'estensione della funzione di transizione** alle parole sull'alfabeto input:

$$\delta^* : Q \times \Sigma^* \rightarrow Q$$

così definita

se $|x| = 0$, allora $\delta^*(q, x) = q$.

se $|x| > 0$, allora $x = ya$, dove $y \in \Sigma^*$ e $a \in \Sigma$,

allora $\delta^*(q, x) = \delta(\delta^*(q, y), a)$.

Nota che $\delta^*(q, a) = \delta(q, a)$ per ogni $a \in \Sigma$.

DEFINIZIONE FORMALE DI ACCETTAZIONE - 2

Una parola $x \in \Sigma^*$ è accettata da un DFA

$M = (Q, \Sigma, \delta, q_0, F)$ se

$$\delta^*(q_0, x) = r \in F.$$

Il **linguaggio** $L(M)$ **accettato** dal DFA M è l'insieme di tutte le parole accettate da M :

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

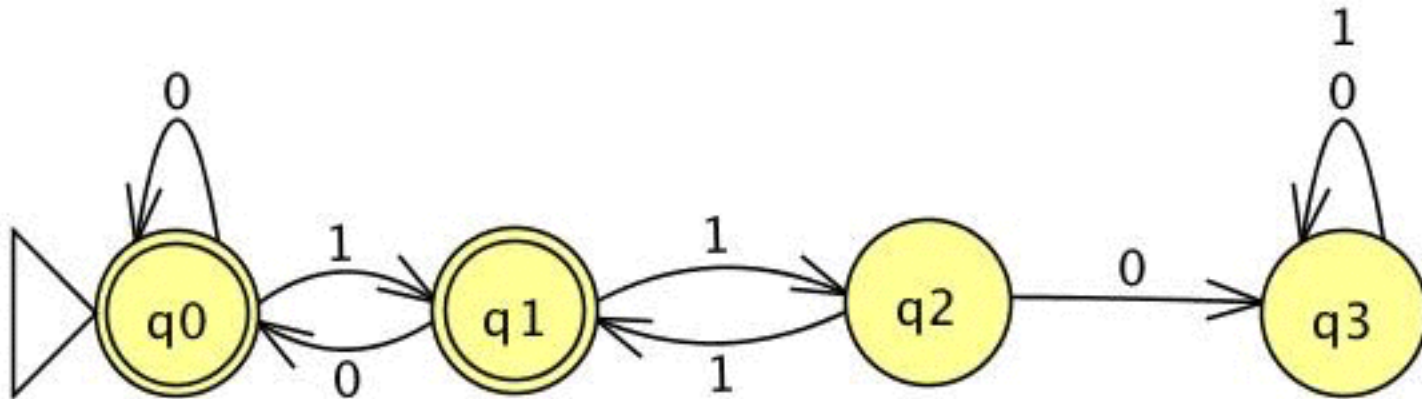
LINGUAGGI REGOLARI

 (DFA) sia la classe dei linguaggi **accettati** da un DFA, formalmente

$$\text{L(DFA)} = \{L \mid \exists M \in \text{DFA e } L(M) = L \}$$

Esempio 3

Si costruisca un DFA che accetta le stringhe binarie, in cui ogni “blocco” di soli 1 ha lunghezza dispari, cioè sotto parole di soli 1 circondate da 0 o terminali



L'algoritmo che decide se un DFA accetta almeno una parola

INPUT: La codifica di $M = (Q, \Sigma, \delta, q_0, F)$.

Fase 1.

Costruisce il grafo $G = (V, E)$ come segue:

1. V è l'insieme Q degli stati
2. Per ogni regola $\delta(q, a) = p$ aggiungi a E , se non è già presente, l'arco (q, p)
3. aggiungi un nuovo vertice t e gli archi (p, t) , per ogni p in F

Usa l'algoritmo $\text{PATH}(G, q_0, t)$ per verificare la presenza di un cammino tra il vertice q_0 e il vertice t in G

L'algoritmo è corretto perché si dimostra che esiste x in $L(M)$ sse c'è un cammino da q_0 a t in G

Un algoritmo alternativo per decidere se DFA accetta almeno una parola

Si dimostra il seguente teorema:

Dato un DFA M con n stati $L(M) \neq \emptyset$ sse esiste una parola di lunghezza minore o uguale a $n-1$ che viene accettata.

Prova.

Dimostriamo che se $L(M) \neq \emptyset$ allora esiste una parola di lunghezza minore o uguale di $n-1$ che viene accettata.

Supponiamo per assurdo che tutte le parole accettate siano di lunghezza almeno n . Prendiamo una parola w tra le più corte parole di lunghezza almeno n accettate da M . Sia $|w| = m$. Il cammino determinato da w sul grafo di transizione, $p_0 \dots p_m$ è lungo $m+1 \geq n+1$ quindi contiene almeno una ripetizione, cioè esistono i e j , con $i < j$, tali che $p_i = p_j$, quindi il cammino $p_0 \dots p_i p_{j+1} \dots p_m$ è più corto di $m+1$. La parola v che etichetta il cammino $p_0 \dots p_i p_{j+1} \dots p_m$ è più corta almeno di 1 di w ed è accettata.

Ma v non può avere lunghezza minore di quella di w , che è tra le più corte accettate e non può essere di lunghezza minore di n per ipotesi. La contraddizione dimostra l'implicazione.

Quindi questo verso del sse è dimostrata, l'altro è banale.

L'algoritmo

L'algoritmo consiste quindi nel controllare se una delle parole di lunghezza minore o uguale a $n-1$ è accettata dal DFA M dato in input, dove n è il numero degli stati.

Se M accetta una di quelle parole possiamo concludere che il linguaggio non è vuoto, altrimenti concludiamo che è vuoto, in virtù del teorema precedente.

Confronto tra i due algoritmi

Il secondo algoritmo ha una complessità esponenziale in n . Poiché l'algoritmo precedente era lineare nella dimensione del grafo è evidente che questo secondo perde nel confronto. (Per esercizio analizzate più in dettaglio la complessità dei due algoritmi)

Inoltre il primo algoritmo è un esempio di riduzione tra problemi (il problema del non vuoto per un DFA viene ridotto al problema dell'esistenza di un cammino tra due vertici in un grafo diretto), mentre il secondo è un esempio di individuazione di un sottoinsieme finito dello spazio delle soluzioni, infinito, la cui esplorazione basta per risolvere il problema.