

# Automati a stati finiti

Dipartimento di Elettronica, Informazione e Bioingegneria  
Politecnico di Milano

21 febbraio 2024

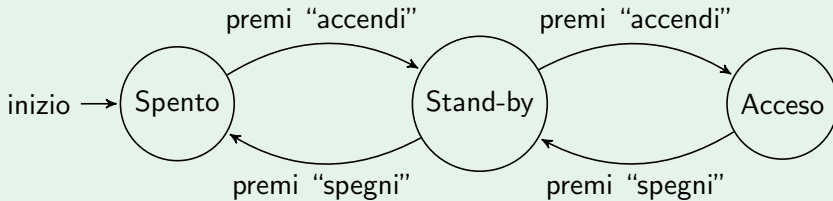
# Modelli operativi

## Un semplice modello di calcolo

- I modelli operativi di calcolo sono definiti come *macchine astratte*
- Modellano il calcolo come una serie di passi *discreti* da una condizione (*stato*) alla successiva
- Il primo (più semplice) che vediamo sono gli **Automati a Stati Finiti** (ASF, o Finite State Automata, FSA)
- Gli FSA hanno memoria del calcolo formata da un insieme **finito** di stati.
- Esempi: {marce di un'automobile}, {fermo, passo, trotto, galoppo}, {in partenza, in viaggio, in arrivo},  $\{1, 2, \dots, k\}$

## Modelli operativi

## Un modello a stati finiti di un televisore



# Formalizzazione

## Costituenti di un FSA

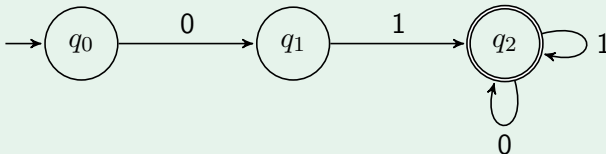
- $\mathbf{Q}$ , l'insieme finito dei suoi stati
- $\mathbf{I}$ , l'insieme finito (alfabeto) dei simboli in ingresso
- $\delta : \mathbf{Q} \times \mathbf{I} \rightarrow \mathbf{Q}$  la funzione di transizione: mappa una coppia (stato corrente, input) in uno stato di destinazione
- Serve definire un inizio della computazione: chiamiamo  $q_0 \in \mathbf{Q}$  lo stato iniziale dell'automa ( $q_0 = \text{Spento}$ , nell'esempio precedente)
  - Si indica con una freccia entrante nello stato iniziale non originata da un altro stato

# FSA Riconoscitore

## Riconoscere un linguaggio con un FSA

- Possiamo usare un FSA per riconoscere le parole di un linguaggio
- Definiamo l'insieme di stati finali  $\mathbf{F} \subseteq \mathbf{Q}$
- Se l'automa, leggendo una stringa, partendo da  $q_0$  termina in uno stato finale, la stringa appartiene al linguaggio

## Riconoscitore per $L = \{\text{stringhe che iniziano con } 01\}$



## Formalizzazione dell'accettazione

### Sequenza di mosse

- Formalizziamo una sequenza di mosse definendo  $\delta^* : \mathbf{Q} \times \mathbf{I}^* \rightarrow \mathbf{Q}$ , estensione di  $\delta$  induttivamente:
  - **Base:**  $\forall q \in \mathbf{Q}, \delta^*(q, \varepsilon) = q$
  - **Passo:**  $\delta^*(q, y.i) = \delta(\delta^*(q, y), i)$ , con  $i \in \mathbf{I}$  e  $y \in \mathbf{I}^*$

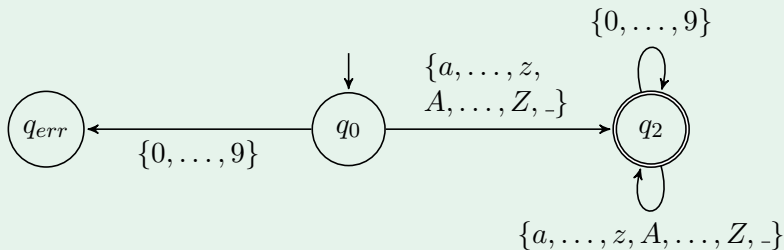
### Accettazione di un linguaggio

- Possiamo formalizzare l'accettazione di un linguaggio  $L$  su  $\mathbf{I}$ , da parte di un FSA  $(\mathbf{Q}, \mathbf{I}, \delta, q_0, \mathbf{F})$  come

$$x \in L \Leftrightarrow \delta^*(q_0, x) \in \mathbf{F}$$

## Riconoscitore degli identificatori del linguaggio C

$L_{id}$  definito su  $I = \{a, b, \dots, z, A, B, \dots, Z, 0, \dots, 9, -\}$

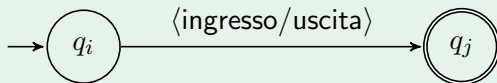


n.b. Arco etichettato con un insieme di  $n$  elementi  $\rightarrow$  abbreviazione per  $n$  archi, uno per ogni simbolo (si possono omettere le graffe se non ambiguo)

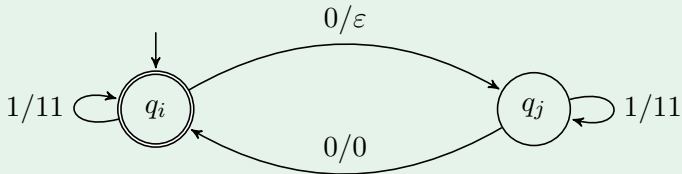
# Traduttori

## FSA traduttore

- Un FSA traduttore da  $L_1$  a  $L_2$  associa un simbolo letto e uno scritto a ogni transizione



$L_1 \subset \{0, 1\}^*$  stringhe con numero di “0” pari,  $\tau$  : dimezza gli “0”, raddoppia gli “1”





# Formalizzazione di un traduttore

## Elementi del traduttore

- Un FSA traduttore: 7-upla  $\mathcal{A} : (\mathbf{Q}, \mathbf{I}, \delta, q_0, \mathbf{F}, \mathbf{O}, \eta)$
- $\langle \mathbf{Q}, \mathbf{I}, \delta, q_0, \mathbf{F} \rangle$  come nell'FSA riconoscitore
- $\mathbf{O}$ : alfabeto di uscita
- $\eta : \mathbf{Q} \times \mathbf{I} \rightarrow \mathbf{O}^*$  funzione di traduzione

## Funzione di traduzione per stringhe $\eta^*$

- $\eta^* : \mathbf{Q} \times \mathbf{I}^* \rightarrow \mathbf{O}^*$  definita in maniera analoga
  - **Base**  $\eta^*(q, \varepsilon) = \varepsilon$
  - **Passo**  $\eta^*(q, y.i) = \eta^*(q, y) \cdot \eta(\delta^*(q, y), i)$ , con  $i \in \mathbf{I}, y \in \mathbf{I}^*$
- L'intero calcolo di traduzione è quindi formalizzato come

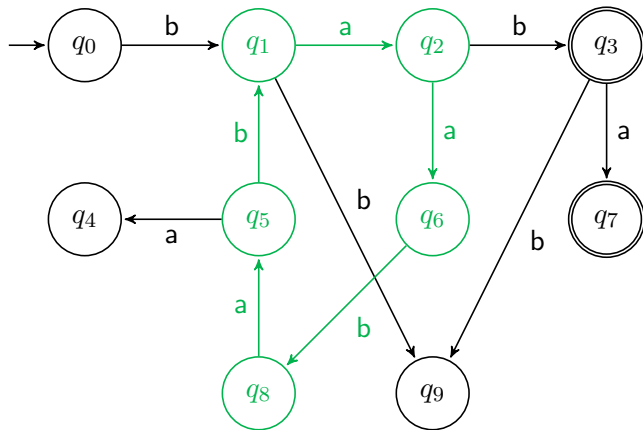
$$\tau(x) = \eta^*(q_0, x) \Leftrightarrow \delta^*(q_0, x) \in \mathbf{F}$$

# Analisi del modello FSA

## Alcune considerazioni

- Modello semplice, ma molto usato
  - E.g.: stati di esecuzione di un processo per il sistema operativo
- Questa semplicità ha un costo?
  - Lo quantificheremo più avanti (alcune calcoli non sono modellabili solo con un FSA)
- Una prima proprietà fondamentale: il *comportamento ciclico* degli FSA

## Esempio di ciclo



Esiste un ciclo  $\delta^*(q_1, aabab) = q_1$  percorribile 0, 1, 2, ...,  $n$  volte

## Formalizzare il comportamento ciclico

### Pumping lemma

- Premessa:  $\exists x \in L$ ,  $L$  riconosciuto da un FSA,  $|x| \geq |Q|$
- Conseguenza: esistono  $q \in Q$ ,  $w \in I^+$  tali che  $x = ywz$  con  $y, z \in I^*$  e  $\delta^*(q, w) = q$ 
  - ovvero esiste una sottostringa di  $x$  che viene riconosciuta dall'automa effettuando un'iterazione su un ciclo di stati
- Dal pumping lemma consegue che  $yw^n z \in L, \forall n \geq 0$ 
  - Segue dal poter effettuare zero o più iterazioni del ciclo
- Da cui “pumping lemma”, posso “gonfiare” il numero di  $w$

# Conseguenze del Pumping lemma

## Proprietà dei linguaggi riconosciuti da FSA

- Posso dire se  $L = \emptyset$ 
  - $\exists x \in L \Rightarrow \exists y \in L, |y| < |Q|$ : se una parola ha “cicli in riconoscimento” li elimino  $\rightarrow$  posso dare in pasto le  $y$  all’FSA (sono finite) e verificare se almeno una  $\in L$
- Posso dire se  $|L| = \infty$ 
  - $\exists x \in L, |Q| \leq |x| < 2|Q|$  implica che  $x$  abbia un ciclo in riconoscimento
- N.B. aver un modo, *in generale* (con altri modelli di calcolo  $\neq$  FSA), di dire se  $x \in L$  non implica saper rispondere queste due domande

# Conseguenze pratiche del Pumping lemma

## Proprietà “utili” di un linguaggio

- Mi interessa aver definito un linguaggio (di programmazione, di descrizione dati) consistente di nessuna parola?
- Mi interessa aver definito un linguaggio di programmazione in cui poter scrivere solo un numero finito di programmi?

## Limitazioni degli FSA

### Riconoscere strutture a parentesi

- $L = \{a^n b^n, n \geq 0\}$  è riconosciuto da un FSA?
- Intuizione: no. Dimostriamolo per assurdo
- sia  $x \in L, x = a^m b^m, \frac{m}{2} > |Q|$ , applicando il pumping lemma abbiamo che  $x = ywz$ , con  $w$  che avere una tra le seguenti forme
  - $w = a^k$ , pompando  $w$  ottengo  $\forall r \in \mathbb{N}, a^{m-k} a^{r \cdot k} b^m \in L, \nexists$
  - $w = b^k$ , pompando  $w$  ottengo  $\forall r \in \mathbb{N}, a^m b^{r \cdot k} b^{m-k} \in L, \nexists$
  - $w = a^k b^h$ , pompando  $w$  ottengo  $\forall r \in \mathbb{N}, a^{m-k} (a^k b^h)^r b^{m-h} \in L, \nexists$

# Limitazioni degli FSA

## Verso modelli più potenti

- Intuitivamente: per “contare” un  $n$  arbitrariamente grande, occorre una quantità di memoria arbitrariamente grande
- Riconoscere strutture a parentesi (HTML, XML, linguaggi di programmazione) non è fattibile da un FSA
- Anche modellare un calcolatore fisico (che è un FSA) come tale può essere scomodo/intrattabile ( $2^{2^{44}}$  stati)
- Sarà necessario “estendere” gli FSA per renderli più efficaci



## Il concetto di chiusura

### Chiusura (algebraica)

- Dati un insieme  $S$ , ed un'operazione definita sui suoi elementi si dice che  $S$  è *chiuso* rispetto all'operazione, se il risultato di applicarla ad un elemento di  $S$  è contenuto in  $S$

### Esempi

- I numeri naturali (insieme  $\mathbb{N}$ ) sono chiusi rispetto alla somma, ma non rispetto alla sottrazione
- L'insieme dei convogli ferroviari è chiuso per concatenazione

# Chiusura nei linguaggi

## Chiusura di famiglie di linguaggi

- Famiglia di linguaggi: un insieme  $\mathbb{L}$  i cui elementi sono linguaggi,  $\mathbf{L} = \{L_i\}$
- $\mathbf{L}$  è chiusa rispetto a un'operazione (binaria)  $\star$  se  $\forall L_1, L_2 \in \mathbf{L}$  vale  $L_1 \star L_2 \in \mathbf{L}$

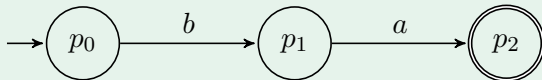
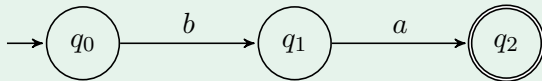
## Linguaggi regolari

- La famiglia di linguaggi riconoscibili con un FSA è la famiglia dei linguaggi *regolari*,  $\mathbf{R}$  o REG
- $\mathbf{R}$  è chiusa rispetto a  $\cup, \cap, \neg, \setminus$ , alla concatenazione, a  $*$  e  $+$

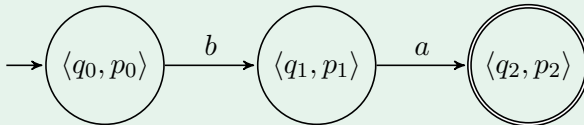
# Intersezione tra linguaggi

## Combinare gli FSA

Dati due FSA riconoscitori

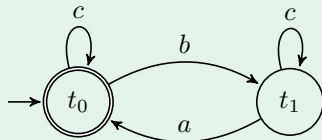
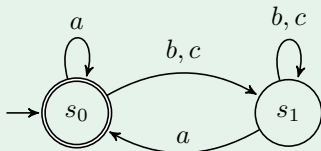


Otengo il riconoscitore del linguaggio intersezione facendoli funzionare "insieme": è possibile una transizione solo se c'è in entrambi

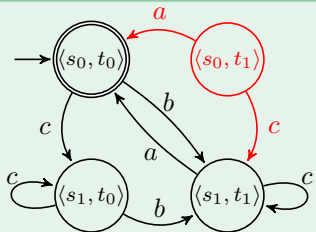


## Intersezione: esempio

### Automi "sorgente"



### Automa intersezione



- Lo stato in **rosso** non è raggiungibile dallo stato iniziale  $\rightarrow$  può essere eliminato
- Con la costruzione a punto fisso a partire da  $\langle s_0, t_0 \rangle$  non viene neppure aggiunto

## Formalizzazione descrittiva

### Chiusura di famiglie di linguaggi

- Dati due automi  $\mathcal{A}_1 : (\mathbf{Q}_1, \mathbf{I}_1, \delta_1, q, \mathbf{F}_1)$ ,  $\mathcal{A}_2 : (\mathbf{Q}_2, \mathbf{I}_2, \delta_2, s, \mathbf{F}_2)$  l'automa intersezione  $\mathcal{A}_\cap = \langle \mathcal{A}_1, \mathcal{A}_2 \rangle$  è dato da
  - Insieme degli stati  $\mathbf{Q}_\cap = \mathbf{Q}_1 \times \mathbf{Q}_2$
  - Alfabeto  $\mathbf{I}_\cap = \mathbf{I}_1 \cap \mathbf{I}_2$
  - Funzione di transizione  $\delta_\cap(\langle q_1, q_2 \rangle, i) = \langle \delta_1(q_1, i), \delta_2(q_2, i) \rangle$
  - Insieme degli stati finali  $\mathbf{F}_\cap = \mathbf{F}_1 \times \mathbf{F}_2$

### Correttezza del riconoscitore: $L(\mathcal{A}_\cap) = L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$

- Si può dimostrare per induzione sulla lunghezza dell'input

# Unione di linguaggi

## Due soluzioni

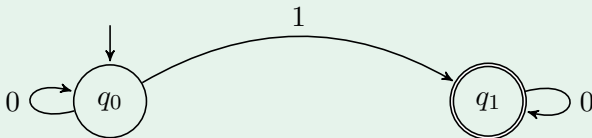
- Una prima soluzione è usare una costruzione analoga all'intersezione, ma che ottiene
  - Insieme degli stati  $\mathbf{Q}_U = (\mathbf{Q}_1 \times \mathbf{Q}_2) \cup \mathbf{Q}_1 \cup \mathbf{Q}_2$
  - Alfabeto  $\mathbf{I}_U = \mathbf{I}_1 \cup \mathbf{I}_2$
  - $\delta_U(\langle q_1, q_2 \rangle, i) = \begin{cases} \langle \delta_1(q_1, i), \delta_2(q_2, i) \rangle & \text{se esistono } \delta_1(q_1, i), \delta_2(q_2, i) \\ \delta_1(q_1, i) \text{ oppure } \delta_2(q_2, i) & \text{altrimenti.} \end{cases}$
  - Insieme degli stati finali  $\mathbf{F}_U = (\mathbf{F}_1 \times \mathbf{F}_2) \cup \mathbf{F}_1 \cup \mathbf{F}_2$
- La seconda soluzione è applicare la legge di De Morgan:  $L_1 \cup L_2 = \neg(\neg L_1 \cap \neg L_2)$  ed usare intersezione e complemento

## Complemento

IL COMPLEMENTO SI FA RISPETTO AD  $A^*$  (CON  $A$  L'ALFABETO)

### Costruito operativamente

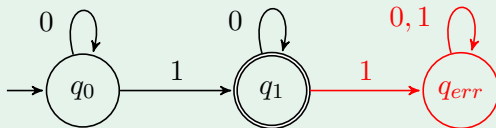
- Idea generale: “rendere finali gli stati non finali e viceversa” ( $F_{compl} = Q \setminus F$ )
- È sufficiente scambiare i ruoli degli stati?



# Complemento

## Gestire $\delta$ parziali

- No.  $\delta$  è una funzione parziale, negli FSA uno stato non accettante è “implicito”: lo stato di errore



- Aggiunto  $q_{err}$  nell'FSA, “scambiare  $\mathbf{F}$  e  $\mathbf{Q} \setminus \mathbf{F}$ ” funziona
- Problema non così facile da risolvere con altri modelli di calcolo
- In generale: calcolare la risposta negativa a un quesito non è equivalente a quella positiva.