

Algoritmi e principi dell'informatica

Matteo Pradella

Email: matteo.pradella@polimi.it

DEIB, edificio 22, piano 3; tel: 3495

ricevimento

Mar. 16.30 (meglio: su appuntamento)

Algoritmi e Principi dell'Informatica

- 2 moduli:
 - 1) Informatica Teorica
 - 2) Algoritmi e Strutture Dati (ex Informatica 3)

Obiettivi e motivazioni

Perché l'informatica “teorica”?

La teoria stimolata dalla pratica:

generalità, rigore, “controllo”

- Comprendere a fondo e in maniera critica i principi dell'informatica (riesame approfondito di concetti elementari)
- Costruire basi solide per comprendere e dominare l'innovazione futura (esempio: multimedialità e modellazione della computazione concorrente)

Il programma (Modulo 1)

- **La modellizzazione informatica**
(Come descrivo un problema e la sua soluzione):
non tanto singoli modelli specialistici
piuttosto creare la capacità di capire e “inventare”
modelli vecchi e nuovi
- **La teoria della computazione:**
che cosa so/posso fare con lo strumento informatico
(quali problemi so risolvere)?

Il programma (Modulo 2)

- **Teoria della complessità**
quanto *costa* risolvere un problema informatico?
- **Complessità di algoritmi e strutture dati**
Algoritmi e strutture dati classiche: ordinamento, pile, liste, code, tabelle hash, alberi e grafi.

L'organizzazione

- Prerequisiti:
 - Le basi essenziali di Informatica
 - Elementi di matematica non continua (Algebra e Logica)
- Lezioni e esercitazioni (stile classico)
 - Interazione auspicata vivamente:
 - Intervento diretto a lezione
 - Ricevimento
 - Email

L'organizzazione (continua)

- Siti internet del corso:
 - <https://martinenghi.faculty.polimi.it/courses/api/index.html>
 - <https://barenghi.faculty.polimi.it/doku.php?id=teaching:api>
 - BeeP (Pradella)
- Esame basato sulla *capacità di applicare, non di ripetere*:
 - principalmente scritto
 - possibilità di consultare testi

L'organizzazione (continua)

- Materiale didattico (Mod. 1):
 - Testo ufficiale:
Mandrioli, Spoletini: *Informatica teorica*, II ed., 2011
 - Eserciziario:
Lavazza, Mandrioli, Morzenti, San Pietro, II ed., Esculapio
 - Mandrioli, Spoletini
Mathematical logic for computer science: an Introduction,
Esculapio
 - Temi d'esame risolti

L'organizzazione (continua)

- Materiale didattico (Mod. 2):

ufficiale: Algoritmi e Principi dell'Informatica,
ISBN 9781307547382, McGraw-Hill 2020 – versione Create.

(libro completo: Cormen T. H., Leiserson C. E., Rivest R. L., Stein
C. Introduzione agli algoritmi e strutture dati, 3/ed)

I modelli in campo ingegneristico

- In ingegneria la fase di progetto si basa sull'uso di *modelli*
- Infatti è spesso impossibile o impraticabile la verifica di soluzioni nel mondo reale
- Talvolta modelli *fisici* (es. ponte o diga in miniatura)
- Spesso modelli *formali*: oggetti matematici che fungono da rappresentazioni astratte di entità reali complesse

Uso dei modelli formali

1. Formalizzazione del problema: dalle entità reali ad astrazione matematica
2. Risoluzione del problema formalizzato
3. Interpretazione dei risultati ottenuti dal modello => valutazione/deduzione delle scelte di progetto

Il modello è *adeguato* se i risultati ottenuti riflettono le proprietà che ci interessano del sistema fisico
(entro limiti di approssimazione accettabili)

I modelli per l'informatica

Fasi principali dell'Ingegneria del SW classica:

1. Analisi dei requisiti => *documento di specifica*
2. Progetto => *architettura sw*
3. Implementazione => *codice*

Tradizionalmente si usa linguaggio naturale o
“pallogrammi” con semantica informale

La tendenza attuale è verso un uso sempre maggiore di
linguaggi formali (o semi-) in tutte le fasi di
produzione del sw

=>

Migliore affidabilità e facilità di manutenzione, ma
soprattutto permette l'uso di **strumenti automatici**

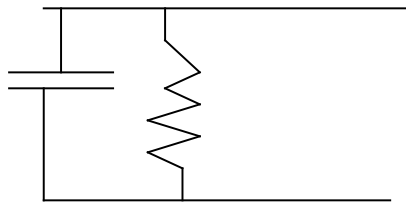
I modelli dell'informatica: qualche nota

- Non (sol)tanto discreti rispetto a continui
(bit e byte rispetto a numeri reali ed equazioni varie)
- Quanto:

- *Generali*:
il sistema informatico nel contesto (spesso) di un sistema più ampio: impianto, organizzazione, sistema “embedded”, ...
- *Flessibili*:
spesso non esiste il “modello già pronto”:
occorre saper adattare modelli esistenti ad esigenze specifiche e impreviste
- esistono molti (troppi) modelli specialistici:
occorre saper studiare/inventare nuovi modelli



- Occorre (maggiore) attitudine dinamica e critica:
 - confronto modello-realtà
 - analisi e sintesi del modello/progetto
- rispetto a:



$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2} = g(x, y, z)$$

- Spesso la vera difficoltà di un problema sta nel ... formularlo!

Che cosa significa:

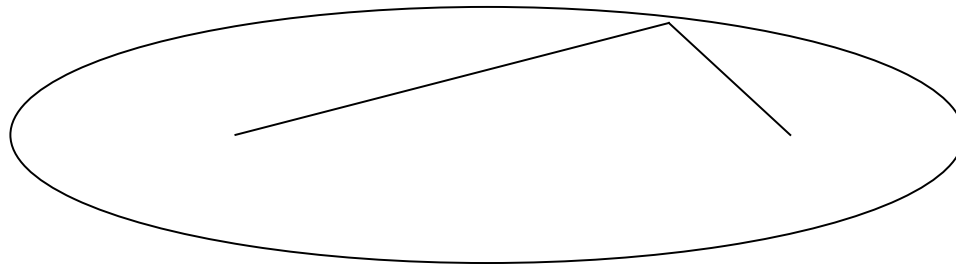
- “automatizzare una procedura d’ufficio”
- “evitare incidenti ferroviari/aerei/...”
- ...

?

- Modelli operazionali
(macchine astratte, sistemi dinamici, ...)
basati sul concetto di stato e di meccanismi
(operazioni) per la sua evoluzione
- Modelli descrittivi
tesi a formulare le proprietà desiderate o temute del
sistema piuttosto del suo funzionamento

Esempi

- Modello operativo dell'ellisse:



- Modello descrittivo dell'ellisse:

$$a x^2 + b y^2 = c$$

- Descrizione operativa dell'ordinamento:
 - Calcola il minimo e mettilo al primo posto;
 - Calcola il minimo degli elementi rimasti e mettilo al secondo posto;
 - ...
- Descrizione non-operativa dell'ordinamento:
 - Individua una permutazione della sequenza data tale che $\forall i, a[i] \leq a[i+1]$

- In realtà le differenze tra modellizzazione operativa e modellizzazione descrittiva non sono così nette: più che altro si tratta di un utile riferimento nel classificare un tipo di modello

Un primo, fondamentale, “meta” modello:
il *linguaggio*

- Italiano, francese, inglese, ...
- C, Pascal, Ada, ...
ma anche:
- Grafica
- Musica
- Multimedialità, ...

Gli elementi di un linguaggio

- Alfabeto o vocabolario
(sinonimi, matematicamente parlando):
Insieme finito di simboli base
 $\{a,b,c, \dots z\}$
 $\{0,1\}$
 $\{\text{Do, Re, Mi, } \dots\}$
 $\{\text{abate, abbaino, } \dots, \text{zuzzurellone}\}$
ASCII
...

- Stringa (su un alfabeto A):
sequenza ordinata e finita di elementi di A , anche con ripetizioni
a, b, aa, alfa, giovanni, alla, nel mezzo del cammin, ...
- Lunghezza di una stringa:
 $|a| = 1, |ab| = 2$
- La stringa nulla ε : $|\varepsilon| = 0$
- A^* = insieme di tutte le stringhe, inclusa ε , su A .
 $A = \{0,1\}, A^* = \{\varepsilon, 0, 1, 00, 01, 10, \dots\}$

- Operazione su stringhe:
concatenazione:
 $x.y$
 $x = abb, y = baba, x.y = abbbaba$
 $x = \text{Quel ramo}, y = \text{del lago di Como},$
 $x.y = \text{Quel ramo del lago di Como}$
“.” : associativa, noncommutativa
- A^* :
monoide libero costruito su A mediante “.”
- ε : elemento neutro rispetto a “.”

Linguaggio

- L sottoinsieme di A^* : $L \subseteq A^*$
Italiano, C, Pascal, ... ma anche:
sequenze di 0 e 1 con numero pari di 1
l'insieme degli spartiti in fa minore
le matrici quadrate il cui determinante è 0
...
- Concetto estremamente ampio, in un certo senso
universale

Operazioni su linguaggi

- Operazioni insiemistiche:

$\cup, \cap, L_1 \cdot L_2, \neg L = A^* - L$ (complemento, a volte \bar{L})

- Concatenazione (tra linguaggi):

$$L_1 \cdot L_2 = \{x.y \mid x \in L_1, y \in L_2\}$$

$$L_1 = \{0,1\}^*, L_2 = \{a,b\}^*$$

$$L_1 \cdot L_2 = \{\varepsilon, 0,1, 0a, 1b, abb, 10ba, \dots \textit{Non ab1!}\}$$

- $L^0 = \{\varepsilon\}, L^i = L^{i-1}.L$
- $L^* = \bigcup_{n=0}^{\infty} L^n$
- NB: $\{\varepsilon\} \neq \emptyset$!
 $\{\varepsilon\} . L = L$;
 $\emptyset . L = \emptyset$
- $+ = \text{“* - 0”}$: $A^+ =$ insieme di tutte le stringhe su A .
 $A = \{0,1\}, A^+ = \{0, 1, 00, 01, 10, \dots\}$
- $L^+ = \bigcup_{n=1}^{\infty} L^n$

Alcuni risvolti “pratici”

- L_1 : insieme dei documenti “Word/Mac”
- L_2 : insieme dei documenti “Word/Windows”
- $L_1 \cap L_2$: insieme dei documenti Word “compatibili Mac e Windows”
- Composizione di un messaggio su rete:
- $x . y . z$:
- x = testata (indirizzo, ...)
- y = testo
- z = “chiusura”

- Il linguaggio: strumento di espressione ...
- di un *problema*
- $x \in L$?
 - Un messaggio è corretto?
 - Un programma è corretto?
 - $y = x^2$?
 - $z = \text{Det}(A)$?
 - Il sonoro di un film è ben sincronizzato con il video?

- $y = \tau(x)$

τ : traduzione: funzione da L_1 a L_2

- τ_1 : raddoppio degli “1” ($1 \dashrightarrow 11$):

$$\tau_1(0010110) = 0011011110, \dots$$

- τ_2 : scambio a con b ($a \dashleftrightarrow b$):

$$\tau_2(\text{abbbaa}) = \text{baaabb}, \dots$$

ma anche:

- compressione di files

- protocolli autocorrettori

- compilazione da linguaggi di alto livello in linguaggi oggetto

- traduzione italiano \dashrightarrow inglese

Conclusione

Il concetto di linguaggio e le operazioni base ad esso associate forniscono un mezzo espressivo estremamente generale per descrivere sistemi di ogni tipo, loro proprietà e problemi ad essi connessi:

- Calcolare il determinante di una matrice;
- Stabilire se un ponte crollerà sotto un certo carico;
-
- In fin dei conti nel calcolatore ogni informazione è una stringa di bit ...

Modelli operazionali

(macchine a stati, sistemi dinamici)

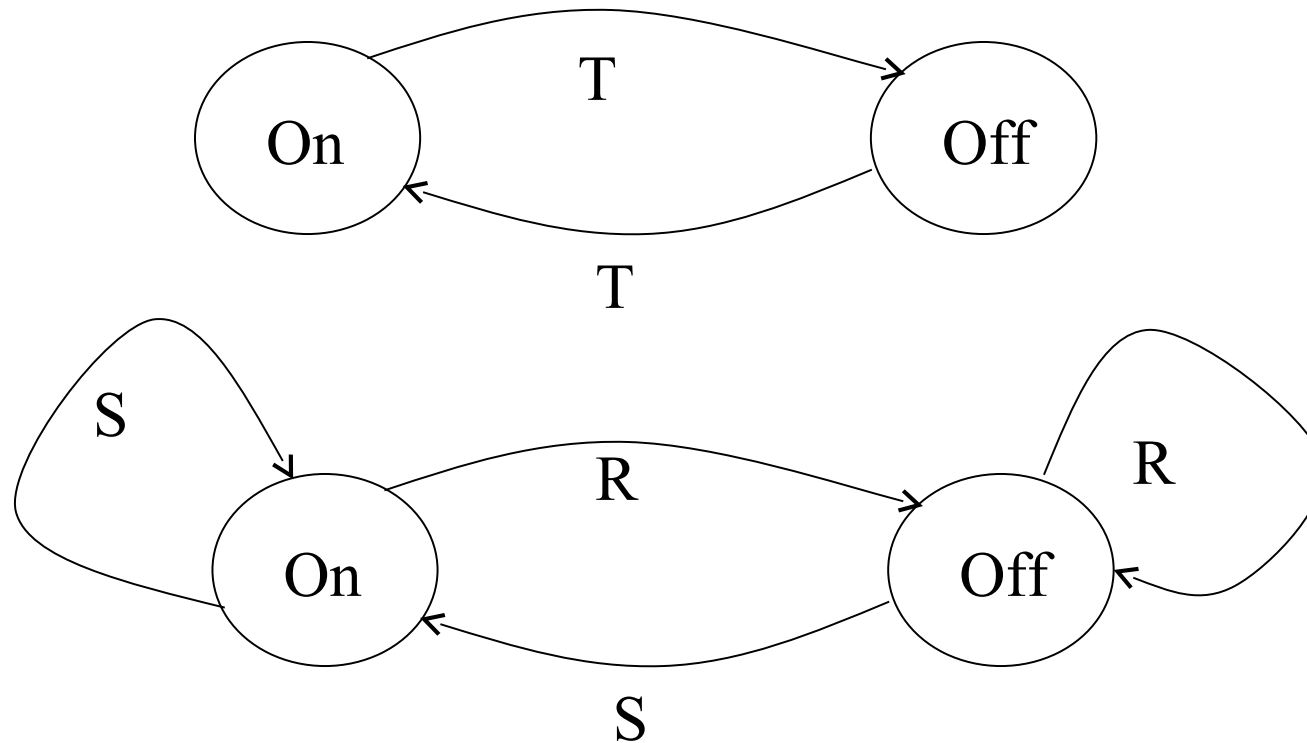
- Le macchine (*automi*) a stati finiti (*FSA*):
 - Un insieme finito di stati:
{Acceso, spento}, {on, off},
{1,2,3,4, ...k}, {canali TV}, {fasce di reddito}, ...

Rappresentazione grafica:



Comandi (ingressi) e transizioni tra stati

- Due semplicissimi flip-flop:



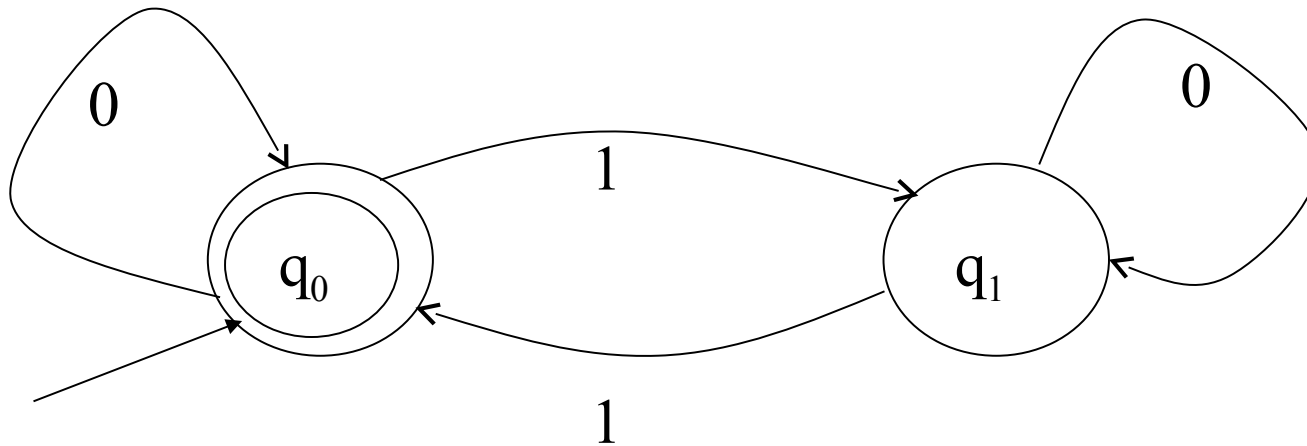
Accensione e spegnimento di luce, ...

Una prima formalizzazione

- Un automa a stati finiti è (costituito da):
 - Un insieme finito di stati: Q
 - Un insieme finito (alfabeto) di ingressi: I
 - Una funzione di transizione (*parziale*):
 $\delta: Q \times I \rightarrow Q$

L'automata come riconoscitore di linguaggi ($x \in L?$)

- Una *sequenza di mosse* parte da uno *stato iniziale* ed è accettata se giunge in uno *stato finale* o di *accettazione*.

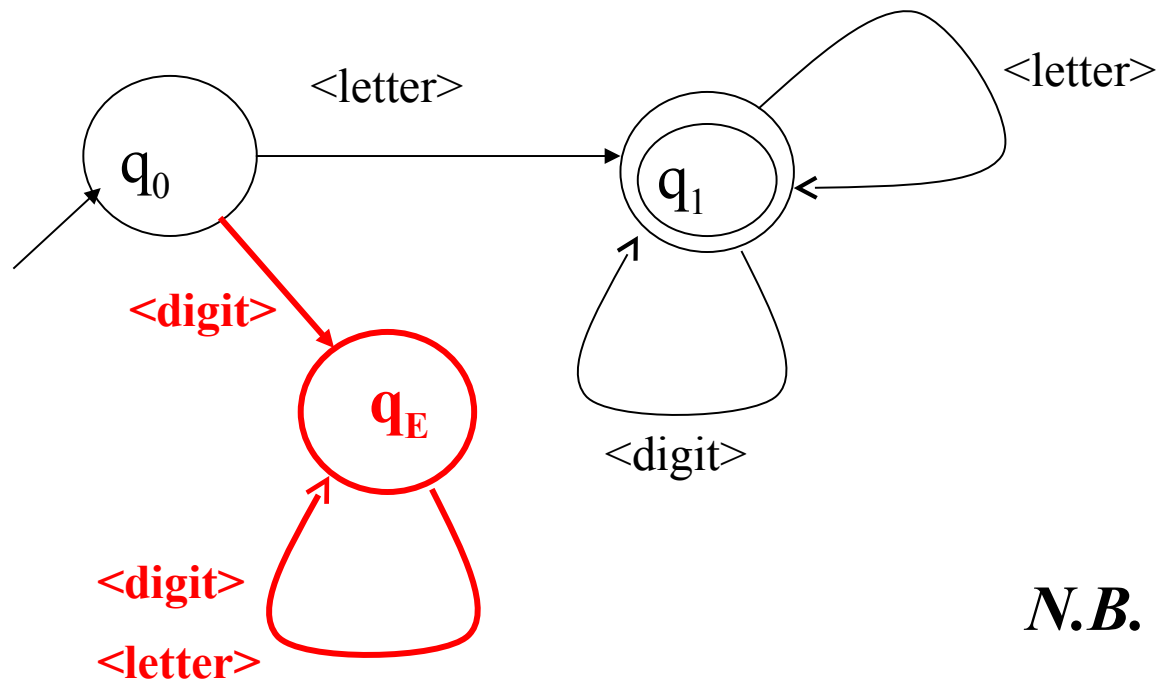


$L = \{\text{stringhe con un numero pari di "1" e un numero qualsiasi di "0"}\}$

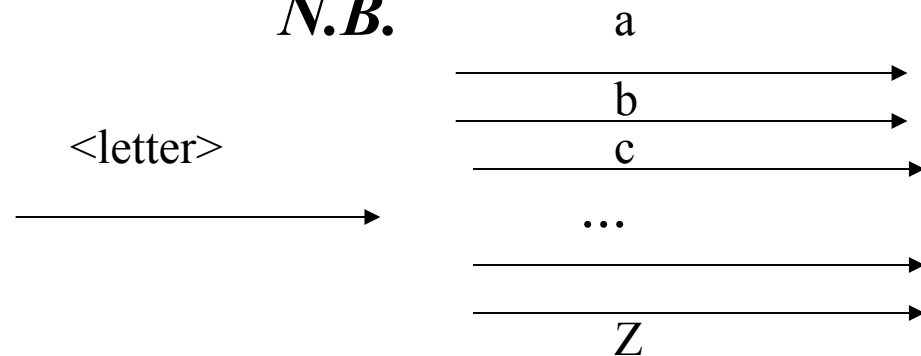
Formalizzazione del riconoscimento di L

- Sequenza di mosse:
 - $\delta^*: Q \times I^* \rightarrow Q$
 δ^* definita induttivamente a partire da δ :
 $\delta^*(q, \varepsilon) = q$
 $\delta^*(q, y.i) = \delta(\delta^*(q, y), i)$
- Stato iniziale: $q_0 \in Q$
- Stati finali o di accettazione: $F \subseteq Q$
- $x \in L \Leftrightarrow \delta^*(q_0, x) \in F$

Riconoscimento di identificatori Pascal



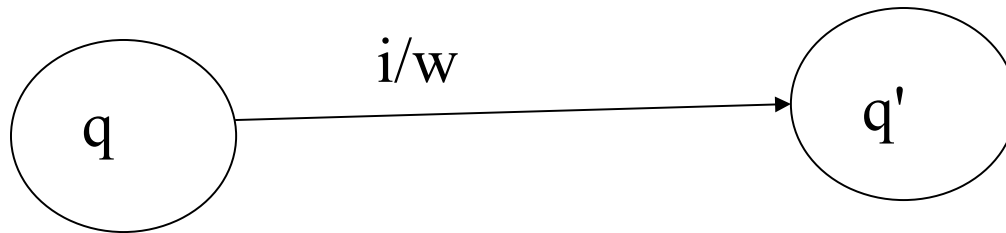
N.B.



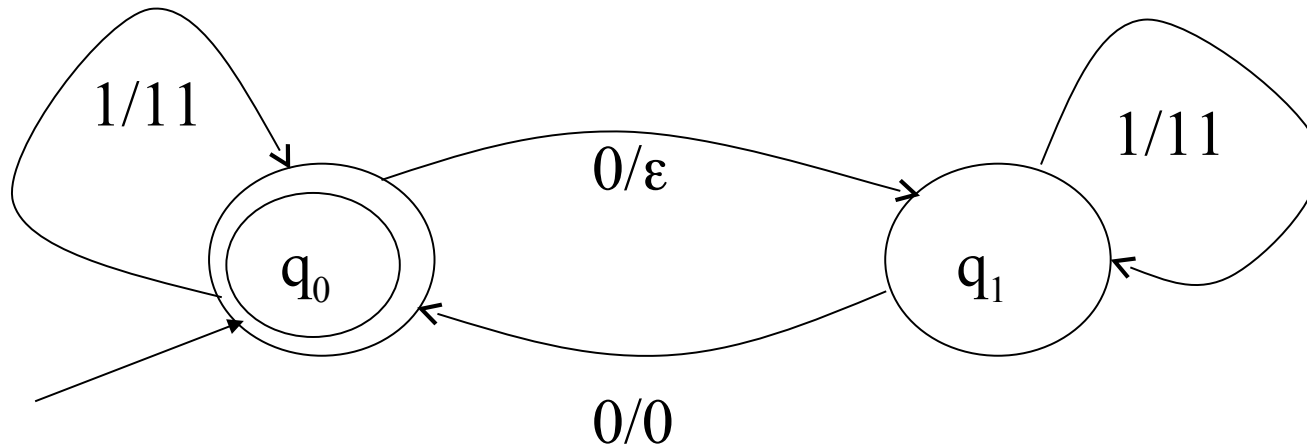
L'automa come traduttore di linguaggi

$$y = \tau(x)$$

Transizione con uscita:



τ : ogni due “0” se ne riscrive 1 e ogni “1” se ne scrivono due (gli “0” devono essere pari)

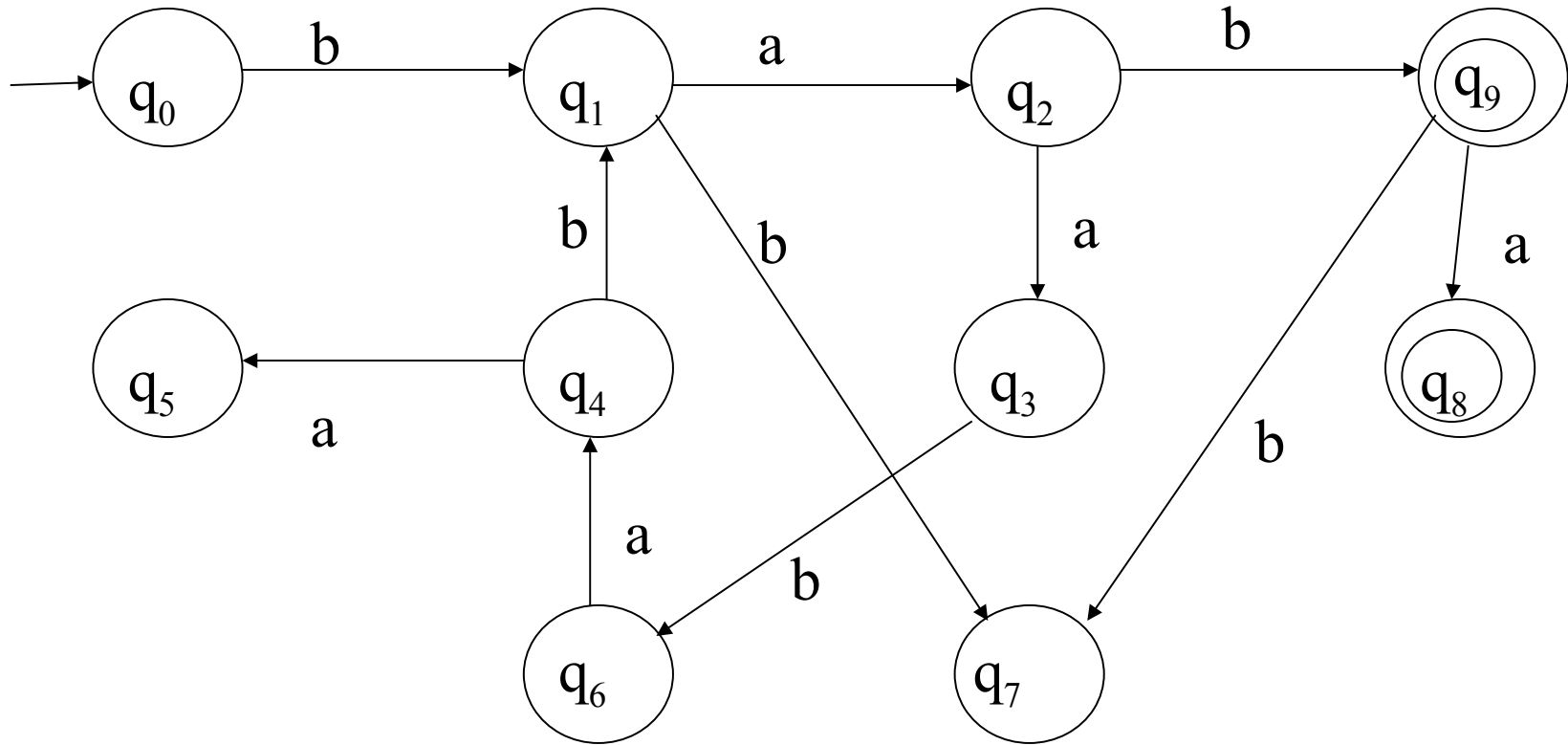


Formalizzazione degli automi traduttori

- $T = \langle Q, I, \delta, q_0, F, O, \eta \rangle$
 - $\langle Q, I, \delta, q_0, F \rangle$: come per A riconoscitore
 - O : alfabeto di uscita
 - $\eta : Q \times I \rightarrow O^*$
- $\eta^* : Q \times I^* \rightarrow O^*$
 - $\eta^*(q, \varepsilon) = \varepsilon$
 - $\eta^*(q, y.i) = \eta^*(q, y). \eta(\delta^*(q, y), i)$
- $\tau(x) = \eta^*(q_0, x)$ sse $\delta^*(q_0, x) \in F$

Analisi del modello a stati finiti (per la sintesi si rimanda ad altri corsi - es. Calcolatori)

- Modello molto semplice ed intuitivo, applicato in molteplici settori, anche fuori dall'informatica
- Si pagherà un prezzo per tale semplicità?
- lo vedremo... (cosa si può e non si può fare)
- Una prima proprietà fondamentale: il *comportamento ciclico* degli automi a stati finiti



C'è un ciclo $q_1 \xrightarrow{aaba} q_1$

Se un ciclo è percorribile una volta, esso è anche percorribile 2, 3, ..., n, ... 0 volte \Longrightarrow

Più formalmente:

- Se $x \in L$ e $|x| > |Q|$ allora
- esistono $q \in Q$ e $w \in I^+$ tali che:
 - $x = ywz$
 - $\delta^*(q, w) = q$
 - inoltre $\forall n \geq 0, yw^n z \in L$

viene detto *Pumping Lemma* (posso “pompare” i w)

Nota: per i dettagli è in generale utile guardare la dimostrazione per esteso sul libro

Dal pumping lemma derivano molte importanti proprietà degli FSA -positive e “negative”-

- $L = \emptyset?$ $\exists x \in L \leftrightarrow \exists y \in L, |y| < |Q|:$
 - numero passi massimo per arrivare a stato finale: $|Q|-1$
- $|L| = \infty?$ $\leftrightarrow \exists x \in L, |Q| \leq |x| < 2|Q|$
 - come sopra, inoltre:
 - lunghezza massima di un ciclo: $|Q|$

Si noti che *in generale*, saper rispondere alla domanda “ $x \in L?$ ” per un generico x , *non* implica saper rispondere alle altre domande! (Con FSA va bene, però non è sempre così)

Alcuni risvolti pratici

- Ci interessa un linguaggio di programmazione consistente di ...
0 programmi corretti?
- Ci interessa un linguaggio di programmazione in cui è possibile scrivere solo un numero finito di programmi?
- ...

Una conseguenza “negativa” del pumping lemma

- Il linguaggio $L = \{a^n b^n \mid n > 0\}$ è riconosciuto da qualche FSA?
- Supponiamo, per assurdo, di sì:
- Consideriamo $x = a^m b^m$, $m > |Q|$ e applichiamo il P.L.
- Casi possibili:
 - $x = ywz$, $w = a^k$, $k > 0 \implies a^{m+r \cdot k} b^m \in L, \forall r : \text{NO}$
 - $x = ywz$, $w = b^k$, $k > 0 \implies \text{idem}$
 - $x = ywz$, $w = a^k b^s$, $k, s > 0 \implies a^{m-k} a^k b^s a^k b^s b^{m-s} \in L : \text{NO}$

- Più intuitivamente: per “contare” n qualsiasi occorre una memoria infinita!
- Rigorosamente parlando ogni calcolatore è un FSA, però si tratta di una astrazione sbagliata: numero di stati intrattabile!
(analog. dinamica di un aereo studiandone ogni singola molecola)
- Passando dall’esempio “giocattolo” $\{a^n b^n\}$ a casi più concreti:
 - Il riconoscimento di strutture parentetiche tipiche dei linguaggi di programmazione non è effettuabile con memoria finita
- Occorrono perciò modelli “più potenti”

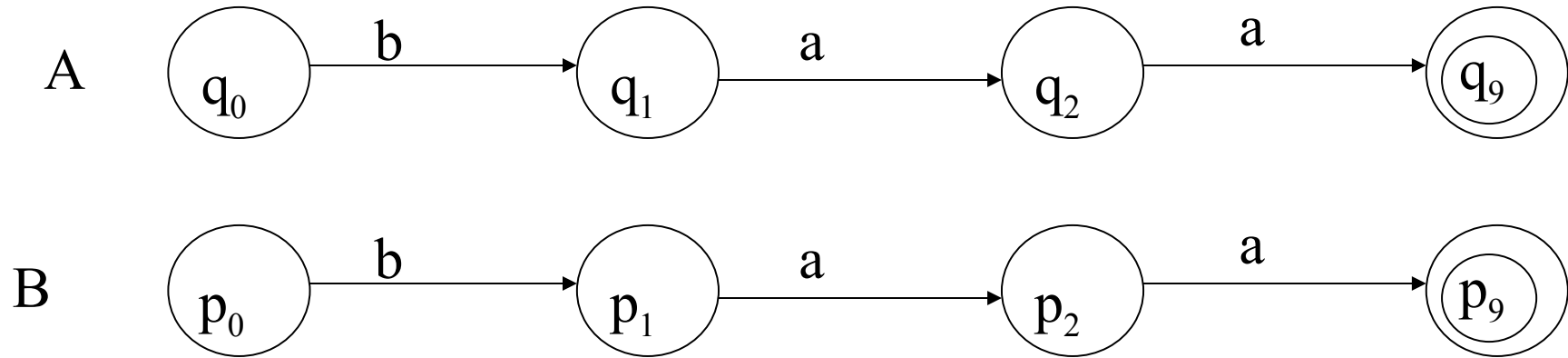
Le proprietà di *chiusura* dei FSA

- Il concetto matematico di chiusura:
 - I numeri naturali sono chiusi rispetto alla somma
 - ma non rispetto alla sottrazione
 - I numeri interi sono chiusi rispetto a somma, sottrazione, moltiplicazione, ma non ...
 - I numeri razionali ...
 - I numeri reali ...
 - Importanza generale del concetto di **chiusura** (di operazioni e relazioni): *insieme più piccolo tale che ...*

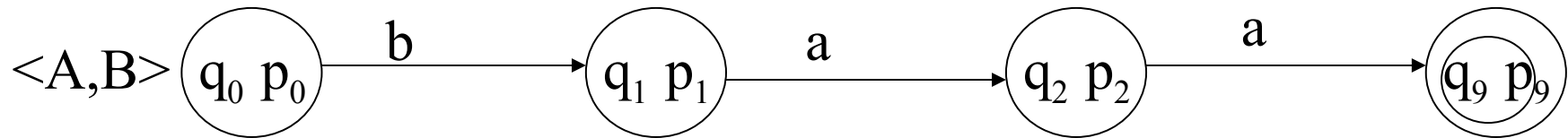
Nel caso dei linguaggi:

- $L = \{L_i\}$: *famiglia* di linguaggi
- L chiusa rispetto a OP se e solo se per ogni $L_1, L_2 \in L$, $L_1 \text{ OP } L_2 \in L$
- **REG**: linguaggi *Regolari*, riconosciuti da FSA
- **REG** chiusa rispetto alle operazioni insiemistiche, alla concatenazione, la “*”, ... e praticamente “tutte” le altre.

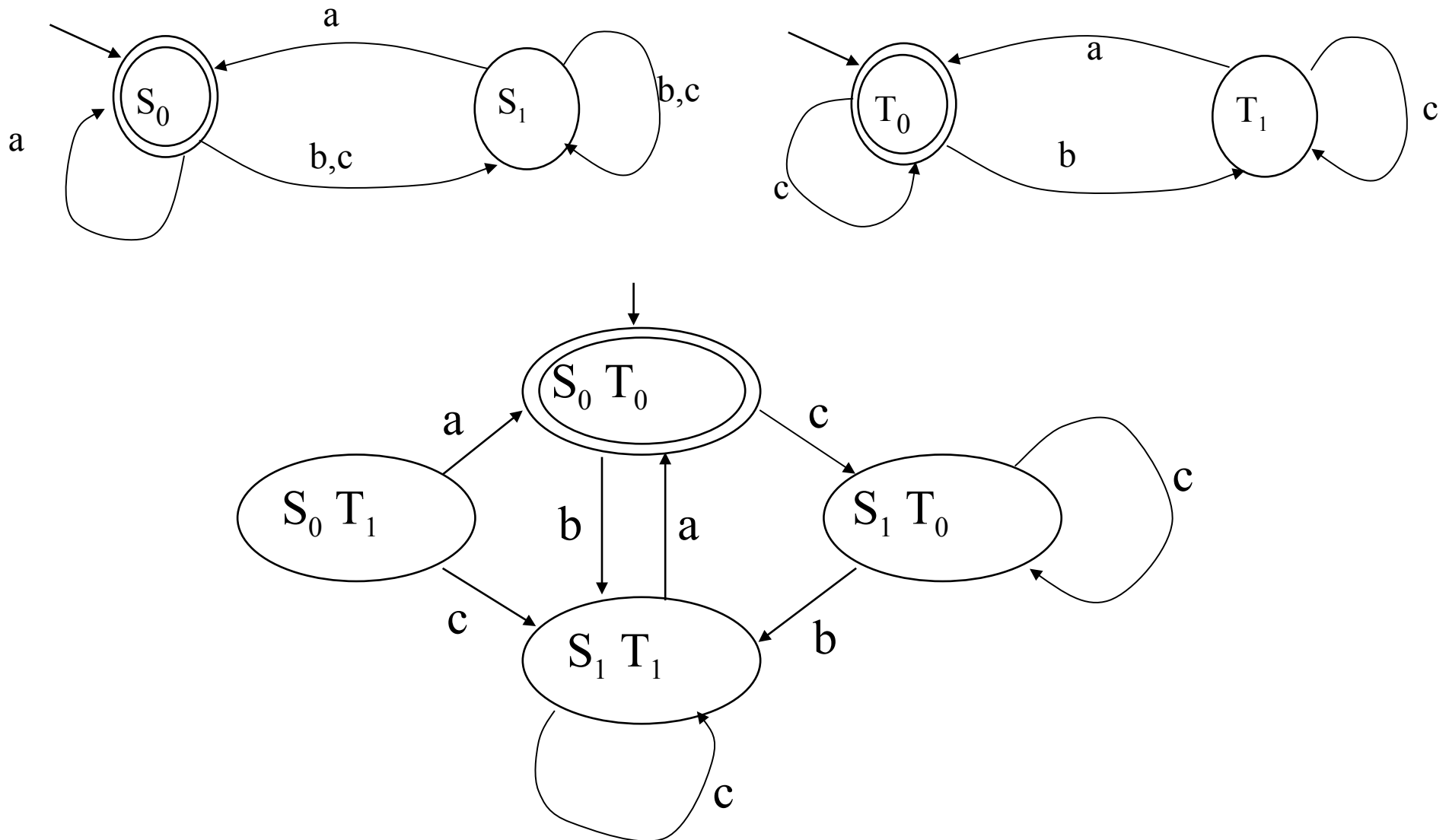
Intersezione



Posso simulare il “funzionamento parallelo” di A e B semplicemente “accoppiandoli”:



Intersezione: esempio



Formalmente:

- Dati $A^1 = \langle Q^1, I, \delta^1, q_0^1, F^1 \rangle$ e
 $A^2 = \langle Q^2, I, \delta^2, q_0^2, F^2 \rangle$
- $\langle A^1, A^2 \rangle =$
 $\langle Q^1 \times Q^2, I, \delta, \langle q_0^1, q_0^2 \rangle, F^1 \times F^2 \rangle$
 $\delta(\langle q^1, q^2 \rangle, i) = \langle \delta^1(q^1, i), \delta^2(q^2, i) \rangle$
- Una semplice induzione dimostra che
 $L(\langle A^1, A^2 \rangle) = L(A^1) \cap L(A^2)$

Possiamo sfruttare la stessa costruzione per l'unione: come?

Unione

- Costruzione concettualmente simile:

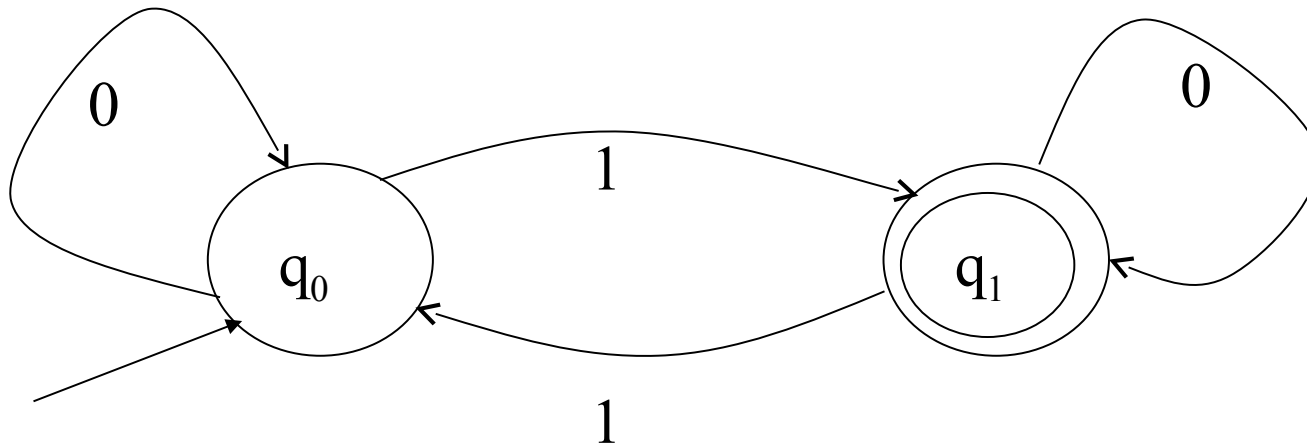
$$\langle Q^1 \times Q^2, I, \delta, \langle q_0^1, q_0^2 \rangle, F^1 \times Q^2 \cup Q^1 \times F^2 \rangle$$

funziona in tutti i casi?

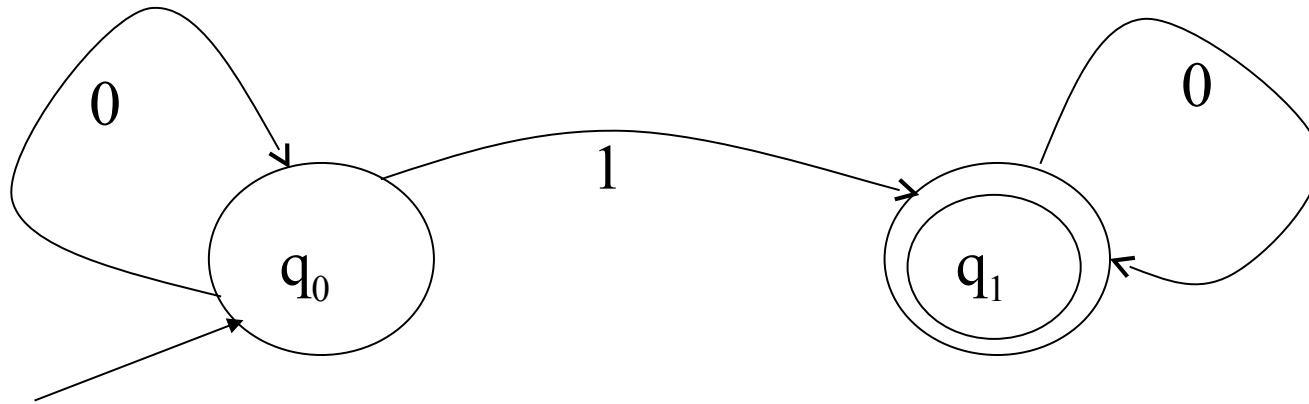
- Un'altra possibilità è sfruttare il complemento e De Morgan:

$$A \cup B = \neg(\neg A \cap \neg B)$$

Complemento:

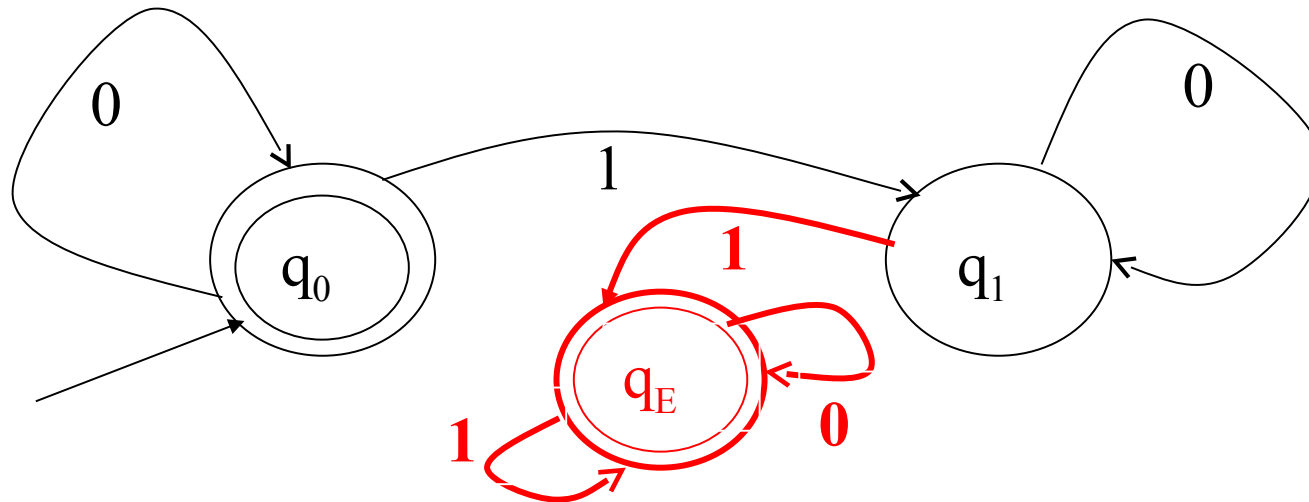


Idea: $F^{\wedge} = Q - F$: Sì però ...



Se mi limito a scambiare F con $Q - F$ cosa succede?

Il problema nasce dal fatto che δ è parziale.

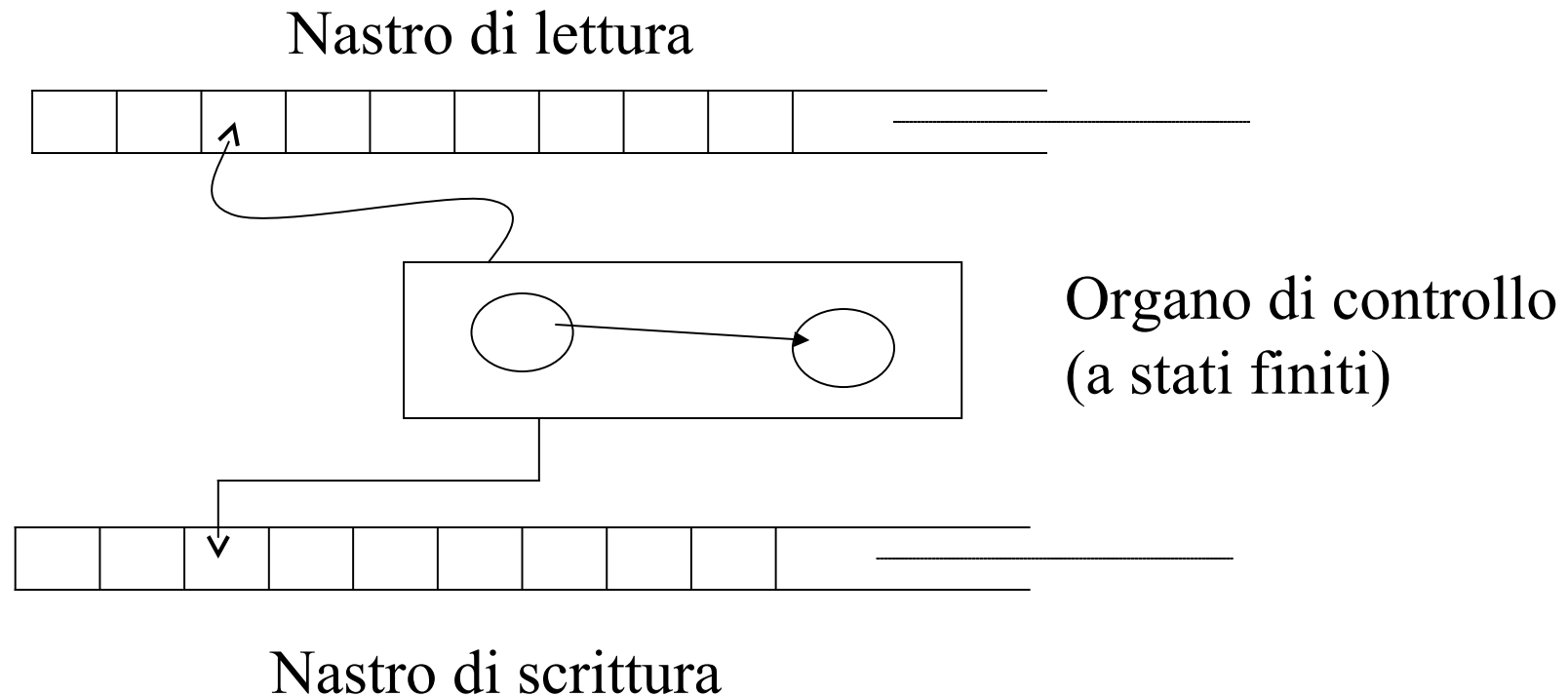


Filosofia generale del complemento

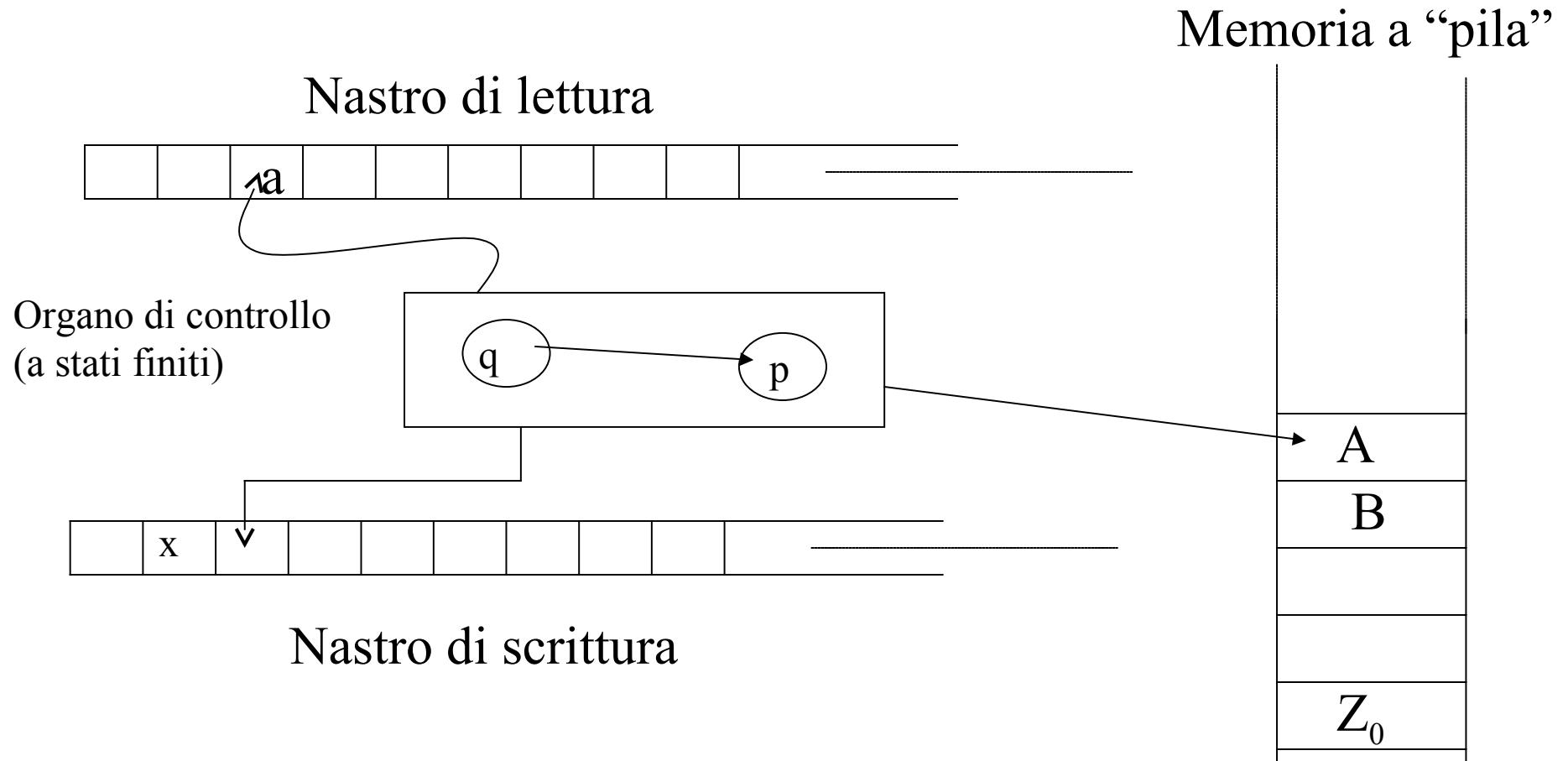
- Se esamino tutta la stringa allora basta “scambiare il sì con il no” (F con Q-F)
- Se però non riesco a giungere in fondo alla stringa (mi “blocco o ...”) allora scambiare F con Q-F non funziona
- Nel caso dei FSA il problema è facilmente risolto ...
- In generale occorre cautela nel considerare la risposta negativa a una domanda come problema equivalente al ricavare la risposta positiva.

Aumentiamo la potenza dei FSA aumentandone la memoria

- Una visione più “meccanica” del FSA:



- Ora “arricchiamolo” un po’:

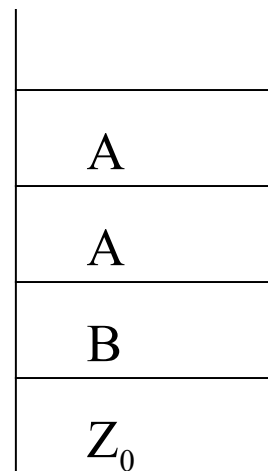
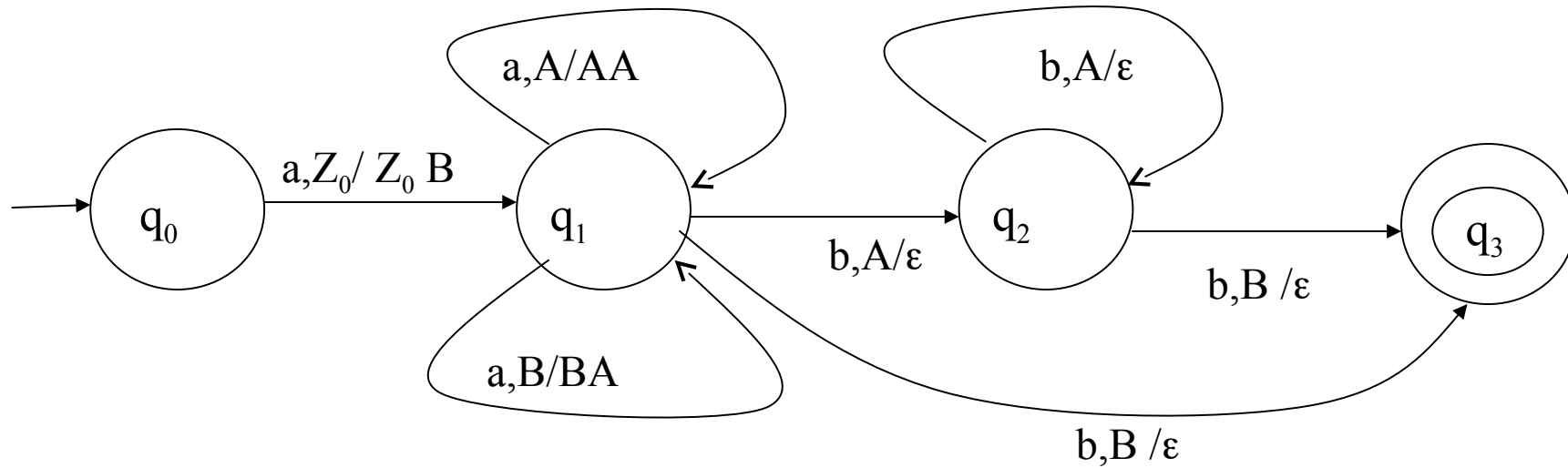


La mossa dell'automata a pila:

- In funzione del:
 - simbolo letto dal nastro di ingresso (però potrebbe anche non leggere nulla ...)
 - simbolo letto dalla pila
 - stato dell'organo di controllo:
 - *cambia stato*
 - *sposta di una posizione la testina di lettura*
 - *sostituisce al simbolo A letto dalla pila una stringa α di simboli (anche nulla)*
 - *(se traduttore) scrive una stringa (anche nulla) nel nastro di uscita (spostando la testina di conseguenza)*

- La stringa di ingresso x viene riconosciuta (accettata) se
 - L'automata la scandisce completamente (la testina di lettura giunge alla fine di x)
 - Giunto alla fine di x esso si trova in uno stato di accettazione (come il FSA)
- Se l'automata è anche traduttore $\tau(x)$ è la stringa che si trova nel nastro di scrittura dopo che x è stata scandita completamente (se x è accettata, altrimenti $\tau(x)$ è indefinita: $\tau(x) = \perp$).
- (\perp : simbolo di “indefinito”)

Un primo esempio: *riconoscimento* di $\{a^n b^n \mid n > 0\}$

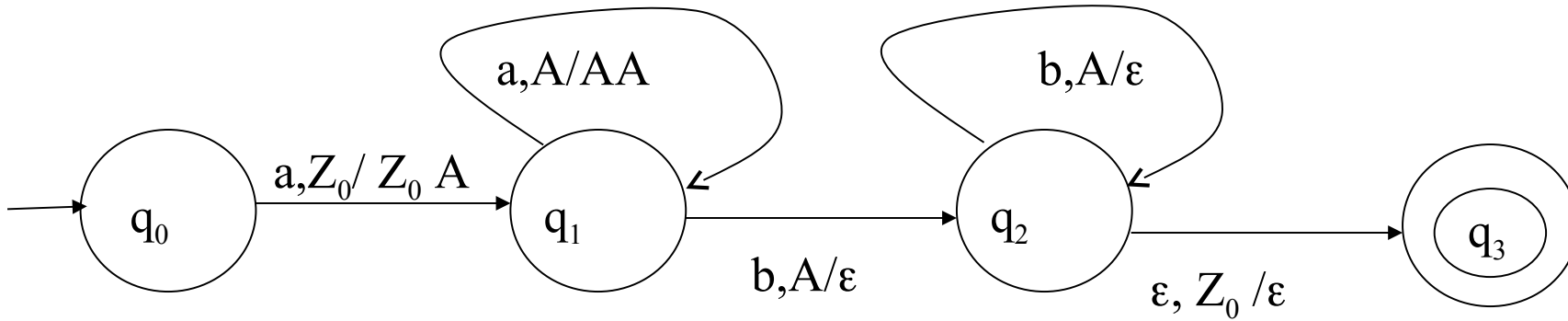


} n - 1

A

uso B per vedere se ho finito

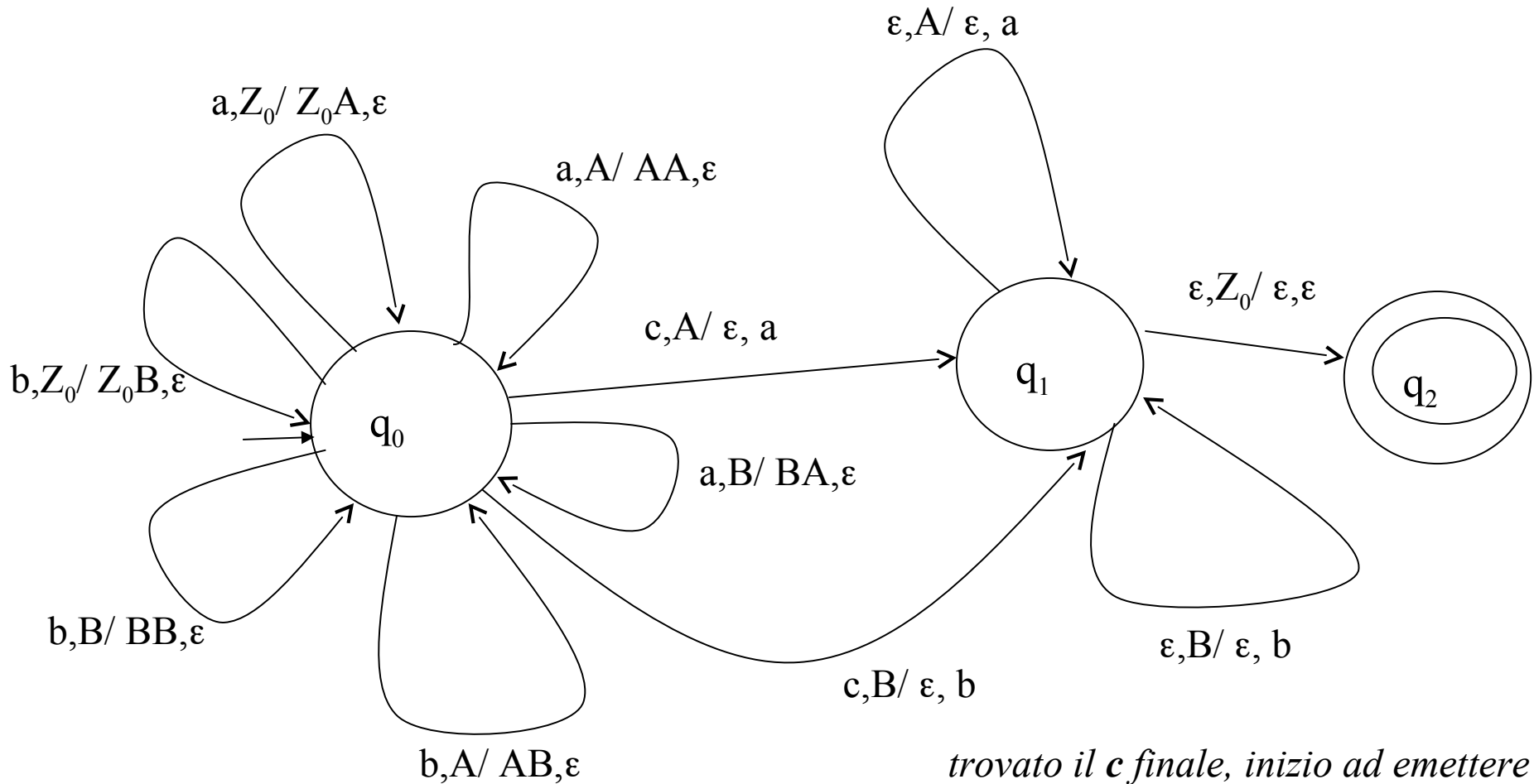
Oppure:



ϵ -mossa!

Non serve più B

Un (classico) automa-traduttore a pila



sulla pila A e B per ogni a e b

*trovato il c finale, inizio ad emettere a e b ,
in base a quello che trovo in pila:
di che traduzione si tratta?*

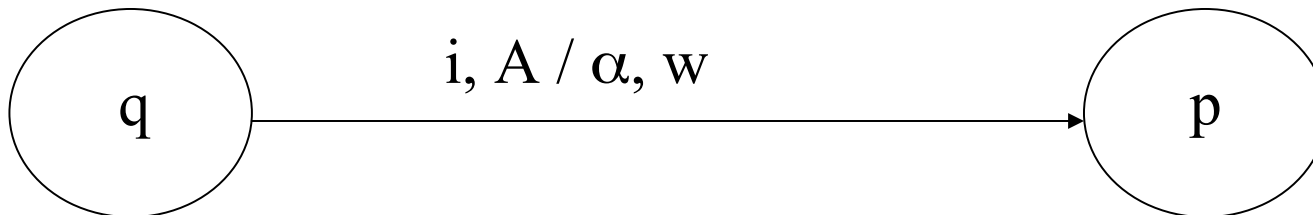
Formalizziamo il funzionamento:

- Automa [traduttore] a Pila: $\langle Q, I, \Gamma, \delta, q_0, Z_0, F, O, \eta \rangle$
- $Q, I, q_0, F \cup \{O\}$ come FSA [T]
- Γ alfabeto di pila (per comodità disgiunto dagli altri)
- Z_0 : simbolo iniziale di pila
- $\delta: Q \times (I \cup \{\varepsilon\}) \times \Gamma \rightarrow Q \times \Gamma^*$ δ : parziale!
- $\eta: Q \times (I \cup \{\varepsilon\}) \times \Gamma \rightarrow O^*$ (η definita dove δ è definita)

Notazione grafica:

$$\langle p, \alpha \rangle = \delta(q, i, A)$$

$$w = \eta(q, i, A)$$



Configurazione (concetto generale di *stato*):

$$c = \langle q, x, \gamma, [z] \rangle:$$

- (indico con [] la parte per un traduttore)
- q : stato dell'organo di controllo
- x : stringa ancora da leggere nel nastro di ingresso (la testina è posizionata sul primo carattere di x)
- γ : stringa dei caratteri in pila
(convenzione: <alto-destra, sinistra-basso>)
- z : stringa già scritta nel nastro di uscita

- Transizione tra configurazioni:

$$c = \langle q, x, \gamma, [z] \rangle \vdash c' = \langle q', x', \gamma', [z'] \rangle$$

$$- \gamma = \beta A$$

Caso 1: $x = i.y$ e $\delta(q, i, A) = \langle q', \alpha \rangle$ (è definita) [$\eta(q, i, A) = w$]

$$- x' = y$$

$$- \gamma' = \beta \alpha$$

$$- [z'] = z.w$$

Caso 2: $x = y$ e $\delta(q, \varepsilon, A) = \langle q', \alpha \rangle$ (è definita) [$\eta(q, \varepsilon, A) = w$]

$$- x' = y$$

$$- \gamma' = \beta \alpha$$

$$- [z'] = z.w$$

- NB: $\forall q, A, \delta(q, \varepsilon, A) \neq \perp \Rightarrow \delta(q, i, A) = \perp \forall i$.
- Altrimenti ... nondeterminismo! (lo vedremo più avanti)

- Accettazione [e traduzione] di una stringa
- \vdash^* : chiusura transitiva e riflessiva di \vdash

$$x \in L [z = \tau(x)]$$

$$\Leftrightarrow$$

$$c_0 = \langle q_0, x, Z_0, [\varepsilon] \rangle \vdash^* c_F = \langle q, \varepsilon, \gamma, [z] \rangle, q \in F$$

Occhio alle ε -mosse, soprattutto a fine stringa!

L'automata a pila in pratica

- Cuore dei compilatori
- Memoria a pila (LIFO) adatta ad analizzare strutture sintattiche “nestate” (espressioni aritmetiche, istruzioni composte, ...)
- Macchina astratta a run-time dei linguaggi con ricorsione
-
Sfruttamento sistematico nel corso di linguaggi e traduttori

Proprietà degli automi a pila (soprattutto come riconoscitori)

- $\{a^n b^n \mid n > 0\}$ riconoscibile da un automa a pila (non da un FSA)

Però $\{a^n b^n c^n \mid n > 0\}$

- NO: dopo aver contato -mediante la pila- n a e “decontato” n b come facciamo a ricordare n per contare i c ?

La pila è una memoria distruttiva: per leggerla occorre distruggerla!

Questa limitazione dell'automato a pila può essere dimostrata formalmente mediante un'estensione del pumping lemma.

Proprietà degli automi a pila (cont.)

- $\{a^n b^n \mid n > 0\}$ riconoscibile da un automa a pila;
 $\{a^n b^{2n} \mid n > 0\}$ riconoscibile da un automa a pila
- Però $\{a^n b^n \mid n > 0\} \cup \{a^n b^{2n} \mid n > 0\} \dots$
 - Ragionamento -intuitivamente- simile al precedente:
 - Se svuoto tutta la pila con n b perdo memoria se ci sono altri b
 - Se ne svuoto solo metà e non trovo più b non posso sapere se effettivamente sono a metà pila
 - La formalizzazione però non è la stessa cosa ...
(piuttosto complicata: non c'è sul libro)

Alcune conseguenze

- ***LP*** = classe dei linguaggi riconosciuti da automi a pila
- ***LP*** non chiusa rispetto all'unione né all'intersezione
- Perché?
- Quanto al complemento ...
Il principio è lo stesso dei FSA: scambiare stati di accettazione con stati di non accettazione.
Nascono però nuove difficoltà

- La δ va completata (come per gli FSA) con lo stato di errore. Occhio però al nondeterminismo causato dalle ε -mosse!
- Le ε -mosse possono causare cicli \rightarrow non si giunge mai in fondo alla stringa \rightarrow la stringa non è accettata, ma non è accettata neanche dall'automa con $F^{\wedge} = Q - F$.
- Esiste però una costruzione che ad ogni automa associa un automa equivalente aciclico (= senza cicli di ε -mosse)
- Non è ancora finita: che succede se si ha una sequenza di ε -mosse a fine scansione con alcuni stati in F e altri no? Cioè:

$$\langle q_1, \varepsilon, \gamma_1 \rangle \vdash \langle q_2, \varepsilon, \gamma_2 \rangle \vdash \langle q_3, \varepsilon, \gamma_3 \rangle \vdash \dots$$

$$q_1 \in F, q_2 \notin F, \dots \quad ?$$

- Occorre “obbligare” l’automa a decidere l’accettazione solo alla fine di una sequenza (necessariamente finita) di ε -mosse.
- Anche questo è possibile mediante apposita costruzione.

Anche in questo caso più che i tecnicismi della costruzione/dimostrazione interessa il meccanismo generale per riconoscere il complemento di un linguaggio: talvolta la stessa macchina che risolve il “problema positivo” può essere impiegata per risolvere anche quello negativo in modo semplice; ma ciò non è sempre banale: occorre la sicurezza di “poter arrivare in fondo” ...

Gli automi a pila [riconoscitori (AP) o traduttori (TP)] sono più potenti di quelli a stati finiti (un FSA è un banale caso particolare di AP; in più gli AP hanno capacità di conteggio illimitato che gli FSA non hanno)
Però anche gli AP/TP hanno i loro limiti ...

... un nuovo e “ultimo” (per noi) automa:

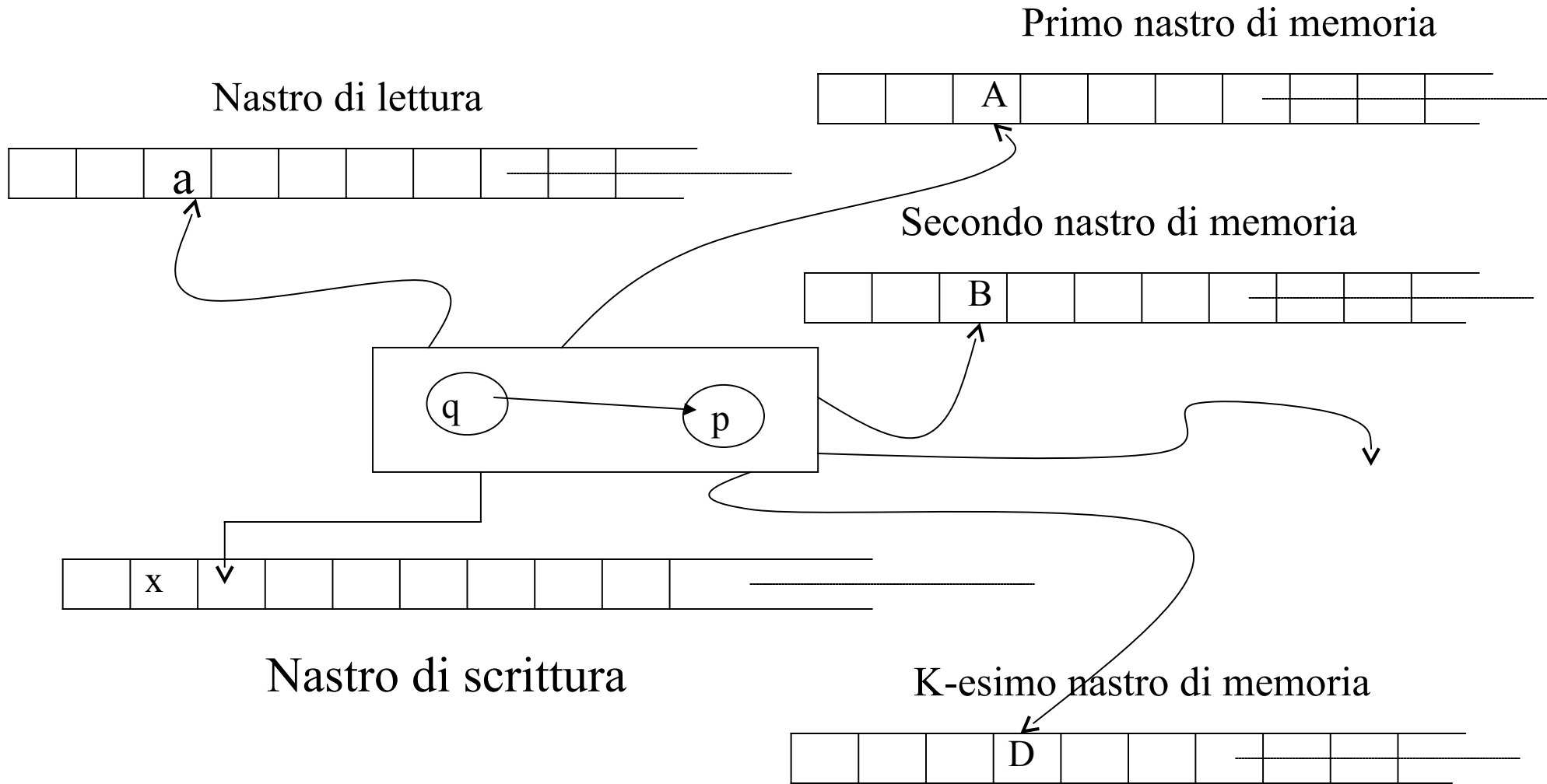
La *Macchina di Turing* (MT) (Alan Turing, 1912-1954)

Modello “storico” di “calcolatore”, nella sua semplicità di notevole importanza concettuale da diversi punti di vista.

Ora lo esaminiamo come automa; successivamente ne ricaveremo proprietà universali del calcolo automatico.

Per ora versione a *k-nastri*, un po’ diversa dal (ancora più semplice) modello originario. Spiegheremo poi il perché di questa scelta.

MT a k-nastri



Descrizione informale e parziale formalizzazione del funzionamento della MT (*formalizzazione completa: esercizio*)

- Stati e alfabeti come per gli altri automi (ingresso, uscita, organo di controllo, alfabeto di memoria)
- Per convenzione storica e convenienza di certe “tecnicità matematiche” i nastri sono rappresentati da sequenze *infinite* di celle $[0,1,2, \dots]$ invece che da stringhe finite. Però esiste un simbolo speciale *blank* (“ “, o \blacksquare “barrato” o “_”) o spazio bianco e si assume che ogni nastro contenga solo un numero finito di celle non contenenti il blank. Evidente l’equivalenza tra i due modi di rappresentare il contenuto dei nastri.
- Testine di lettura/scrittura, simili alle altre testine

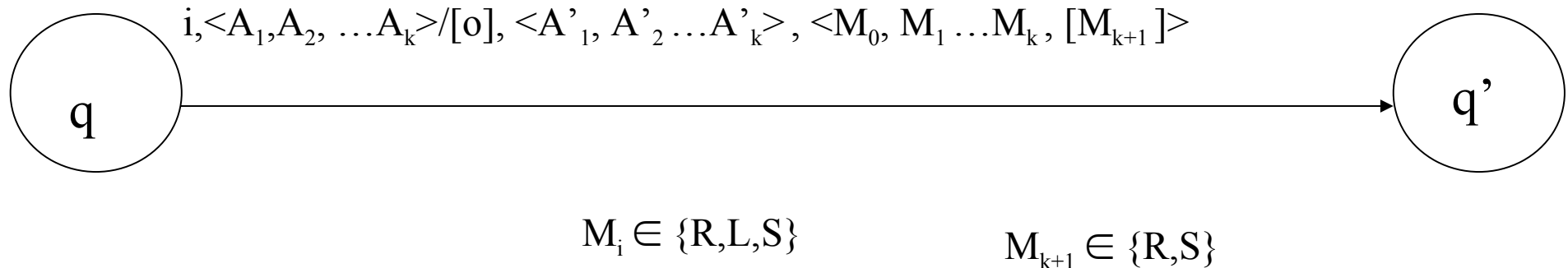
- La mossa della macchina di Turing:
- Lettura:
 - carattere in corrispondenza della testina del nastro di ingresso
 - k caratteri in corrispondenza delle testine dei nastri di memoria
 - stato dell'organo di controllo
- Azione conseguente:
 - cambiamento di stato: $q \rightarrow q'$
 - riscrittura di un carattere al posto di quello letto su ogni nastro di memoria:
 $A_i \rightarrow A_i', 1 \leq i \leq k$
 - [scrittura di un carattere sul nastro di uscita]
 - spostamento delle $k + 2$ testine:
 - le testine di memoria *e di ingresso* possono spostarsi di una posizione a destra (R) o a sinistra (L) o stare ferme (S)
 - la testina del nastro di uscita può spostarsi di una posizione a destra (R) o stare ferma (S) (se ha scritto "e' bene" che si sposti; se si sposta senza aver scritto lascia il blank)

Di conseguenza:

$$\delta, [\eta] : Q \times I \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{R,L,S\}^{k+1} [\times O \times \{R, S\}]$$

(parziali!)

Notazione grafica:



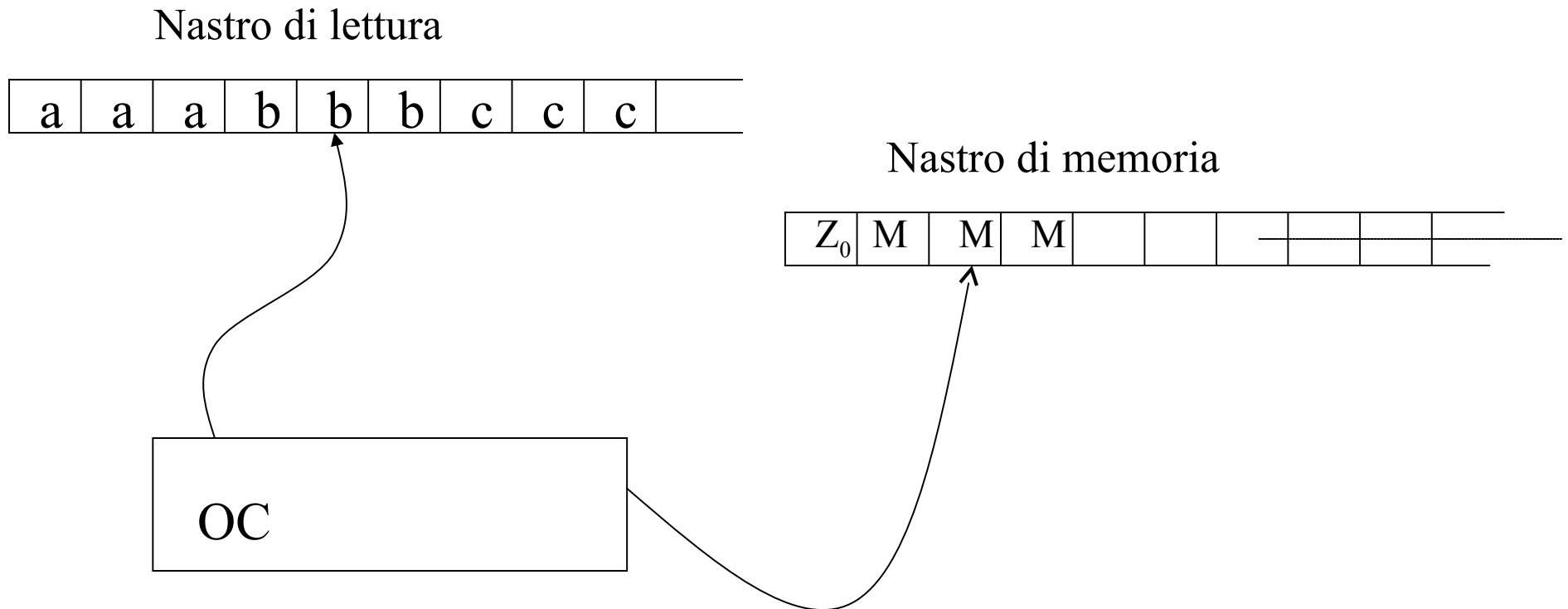
Perché non si perde generalità usando O invece che O^* in uscita?

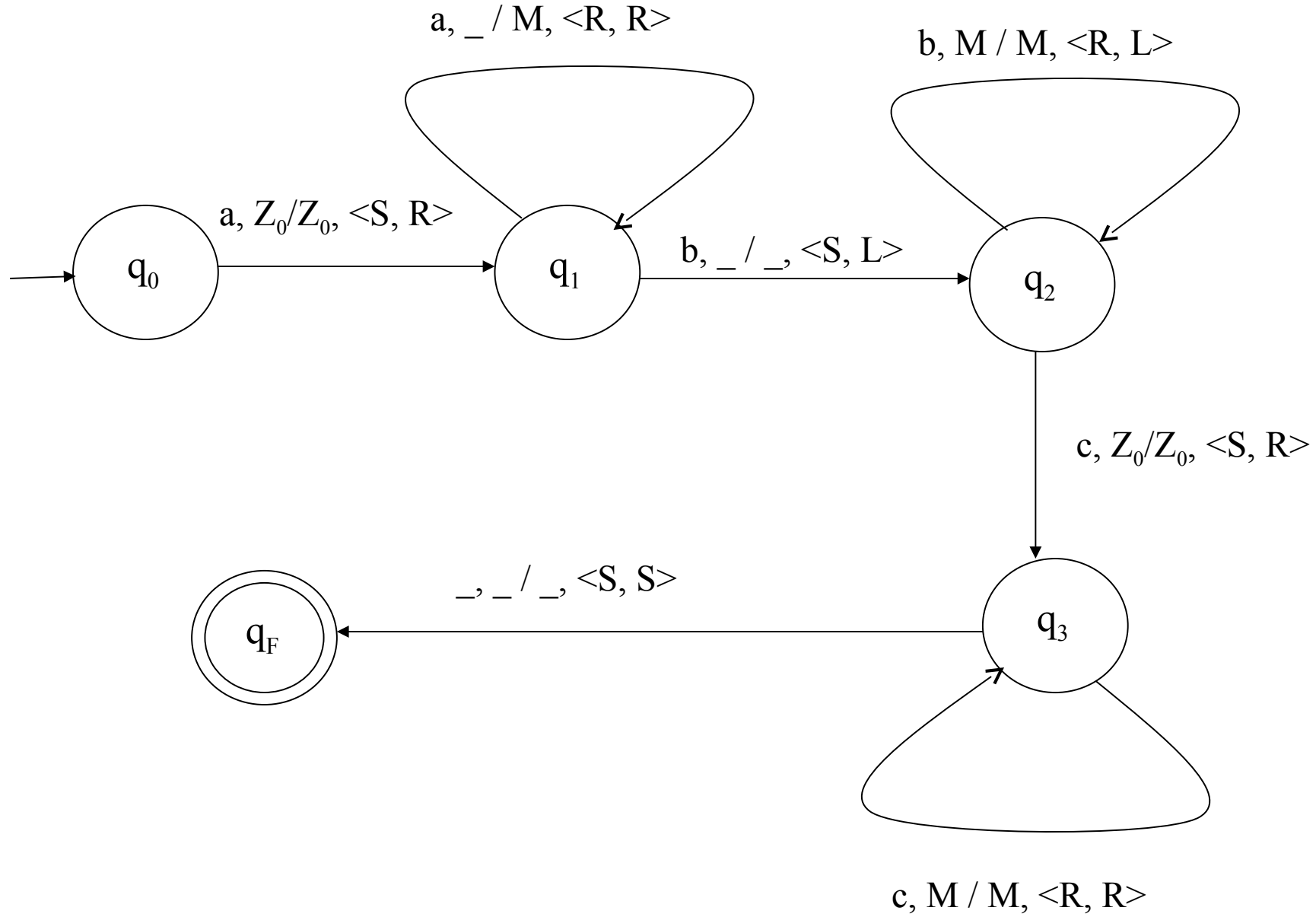
- Configurazione iniziale:
 - Z_0 seguito da tutti blank nei nastri di memoria
 - [nastro di uscita tutto blank]
 - Testine nelle posizioni 0-esime di ogni nastro
 - Stato iniziale dell'organo di controllo q_0
 - Stringa di ingresso x a partire dalla 0-esima cella del nastro corrispondente, seguita da tutti blank

- Configurazione finale:
 - Stati di accettazione $F \subseteq Q$
 - Per comodità, convenzione:
 $\delta, [\eta] (q, \dots) = \perp \forall q \in F$:
 - La macchina si ferma quando $\delta, [\eta] (q, \dots) = \perp$
 - La stringa x di ingresso è accettata se e solo se:
 - dopo un numero finito di mosse la macchina si ferma (si trova in una configurazione in cui $\delta, [\eta] (q, \dots) = \perp$)
 - lo stato q in cui si trova quando si ferma $\in F$
- NB:
 - x *non* è accettata se:
 - la macchina si ferma in uno stato $\notin F$; oppure
 - la macchina non si ferma
 - C'è una somiglianza con l'AP (anche l'AP non loop-free potrebbe non accettare per “non fermata”), però... esiste la MT loop-free?

Alcuni esempi

- MT che riconosce $\{a^n b^n c^n \mid n > 0\}$





Calcolo del successore di un numero codificato in cifre decimali

- M copia tutte le cifre di n su T_1 , alla destra di Z_0 .
Così facendo sposta la testina di T_2 dello stesso numero di posizioni.
- M scandisce le cifre di T_1 da destra a sinistra. Scrive in T_2 da destra a sinistra modificando opportunamente le cifre (i 9 diventano 0, la prima cifra $\neq 9$ diventa la cifra successiva, poi tutte le altre vengono copiate uguali, ...)
- M ricopia T_2 sul nastro di uscita.

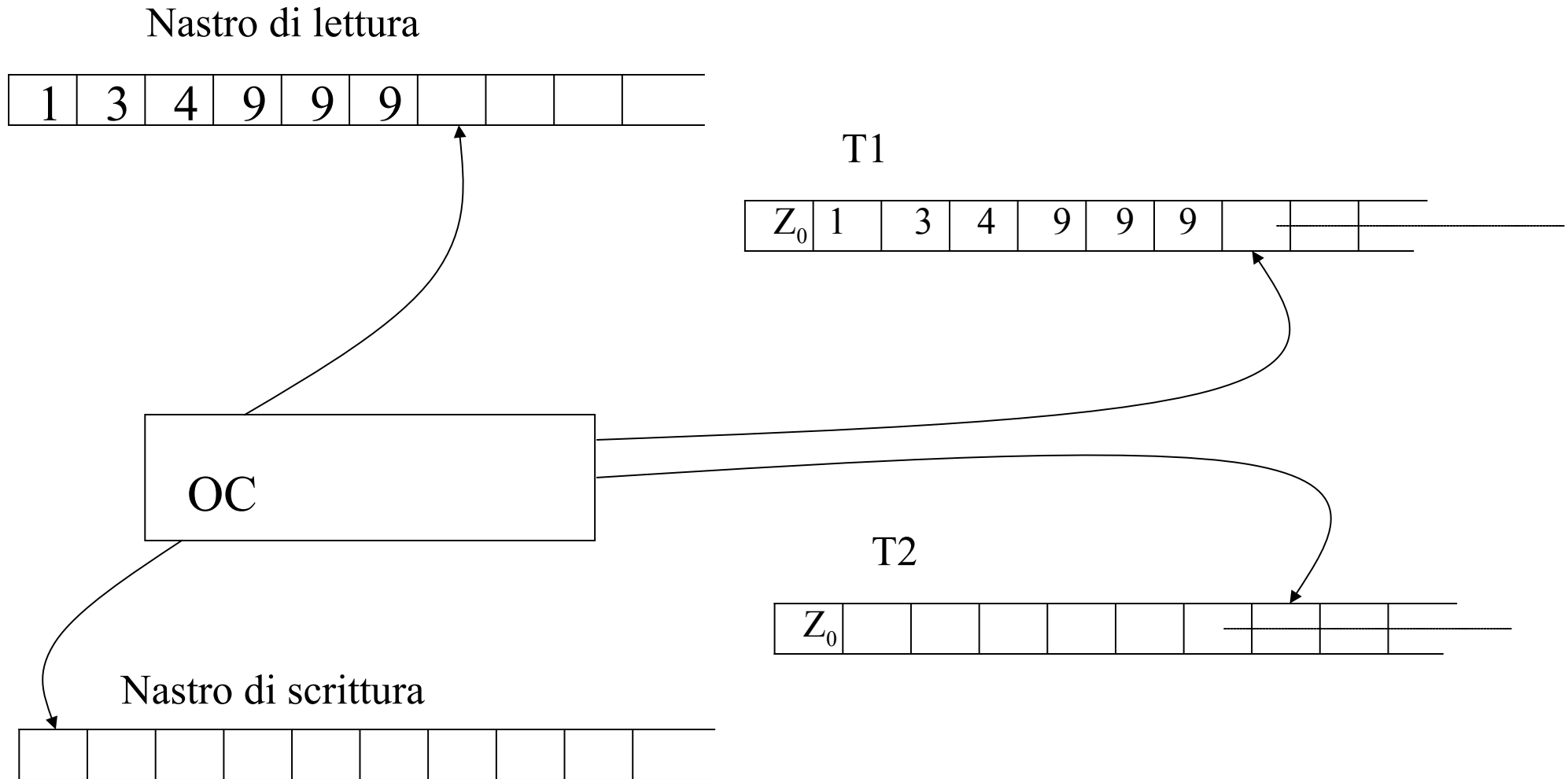
Notazione:

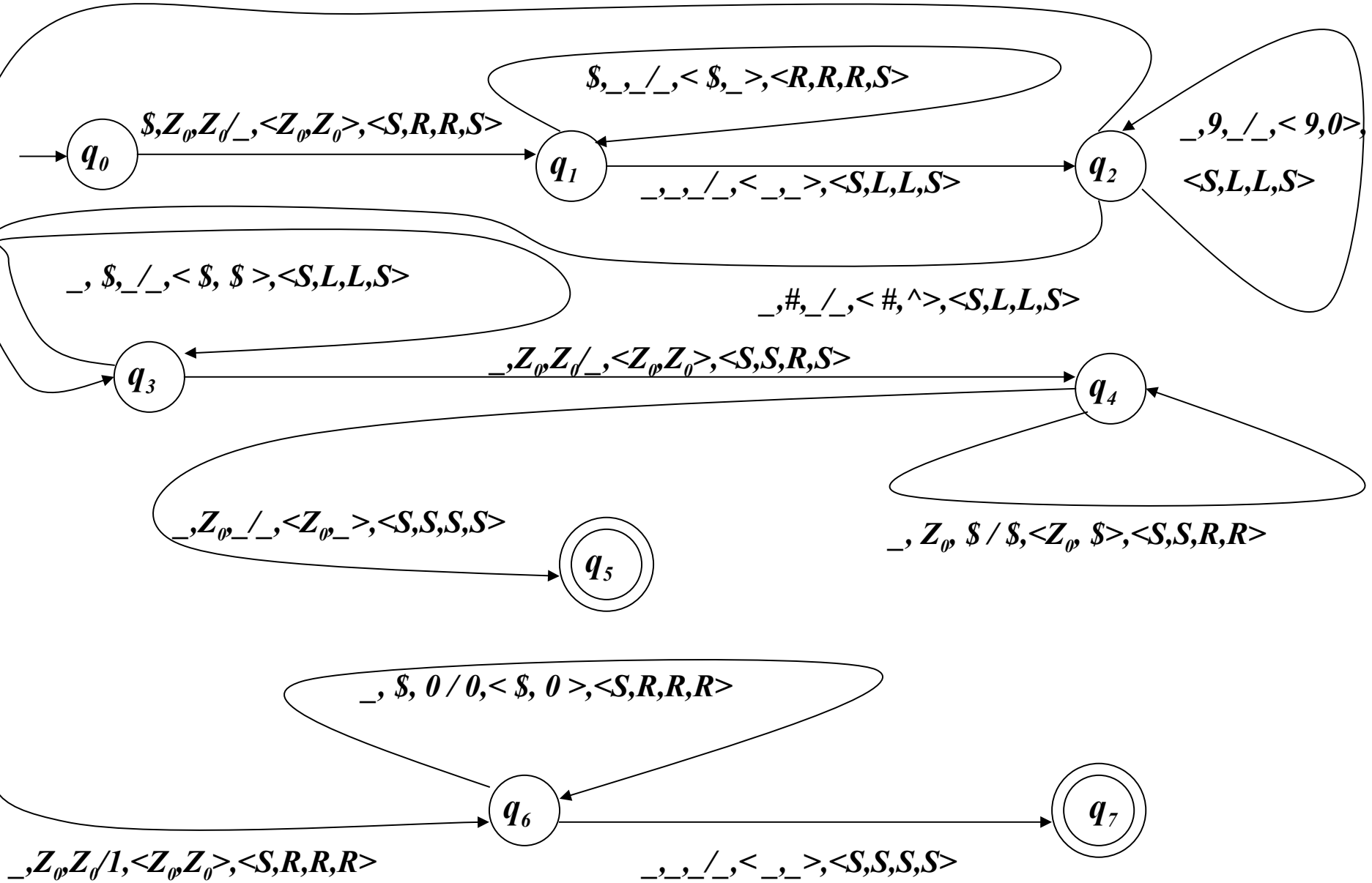
$\$$: *qualsiasi cifra decimale*

$_$: *blank*

$\#$: *qualsiasi cifra $\neq 9$*

\wedge : *il successore della cifra denotata da $\#$ (nella stessa transizione)*





Proprietà di chiusura delle MT

- \cap : OK (una MT puo, facilmente simularne due, sia “in serie” che “in parallelo”)
- \cup : OK (idem)
- Idem per altre operazioni (concatenazione, *,)
- E il complemento?

Risposta negativa! (Dimostrazione in seguito)

Certo se esistessero MT loop-free come gli AP, sarebbe facile: basterebbe definire l'insieme degli stati di halt (facile renderlo disgiunto dagli stati non di halt) e partizionarlo in stati di accettazione e stati di non accettazione.

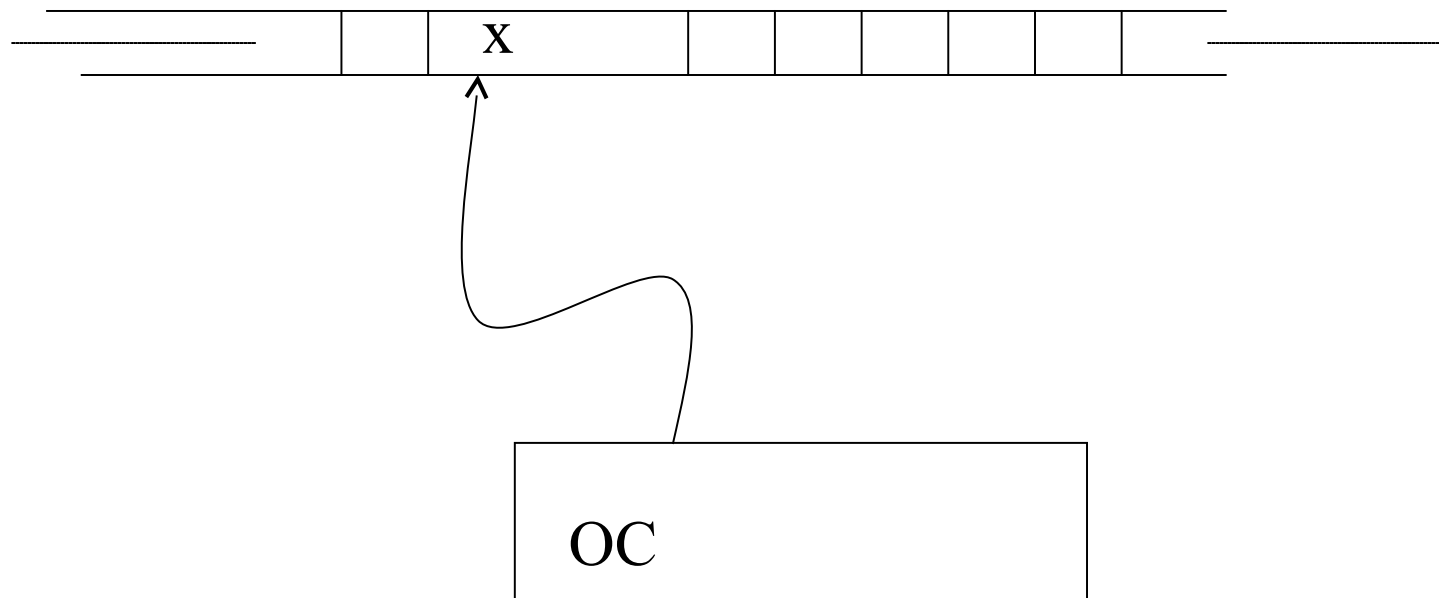
====>

Evidentemente il problema sta nelle *computazioni che non terminano*

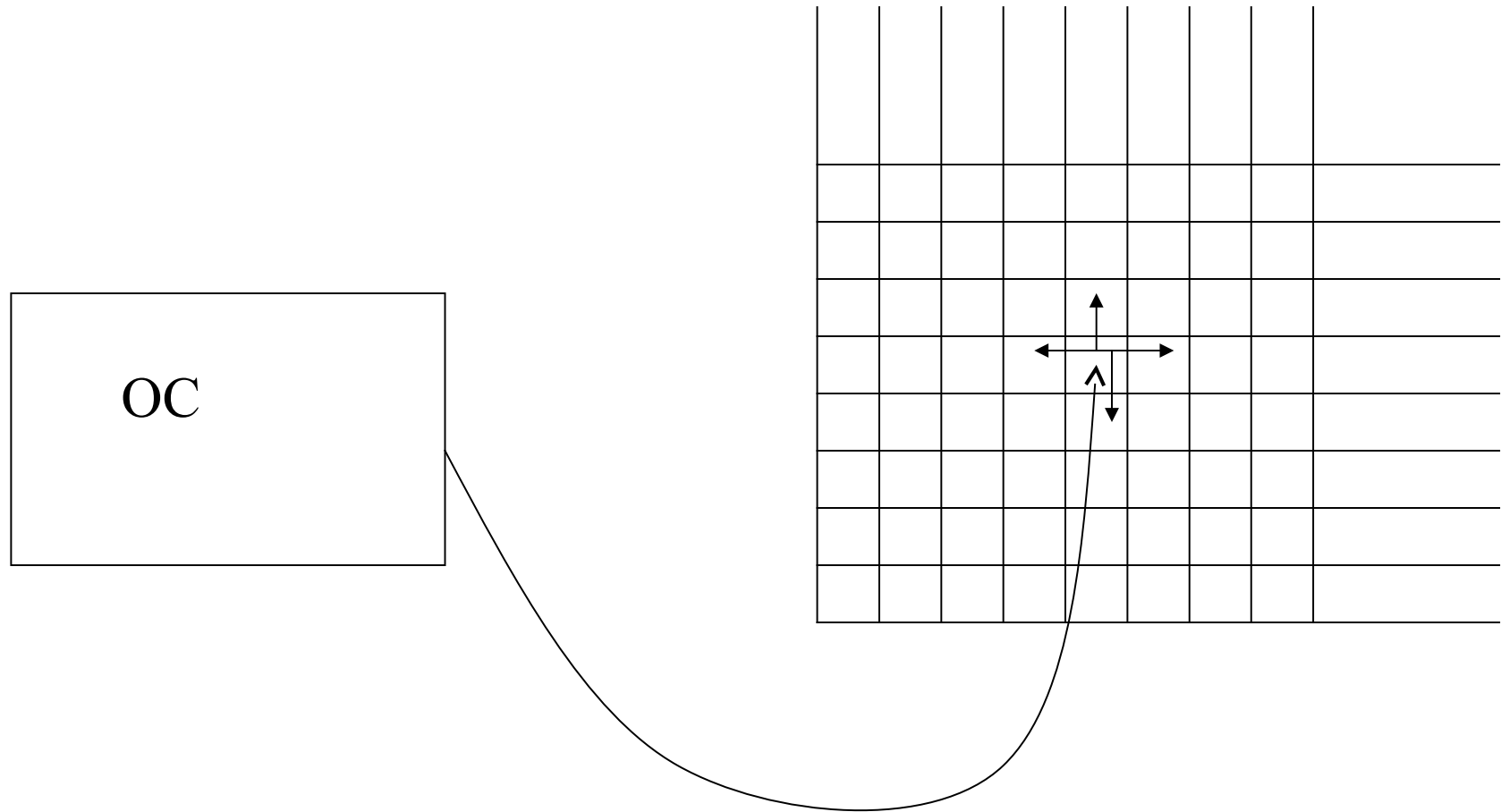
Modelli equivalenti di MT

- MT a nastro singolo (\neq da MT a un nastro - di memoria!)

Nastro unico (di solito illimitato a destra e sinistra):
funge da ingresso, memoria e uscita



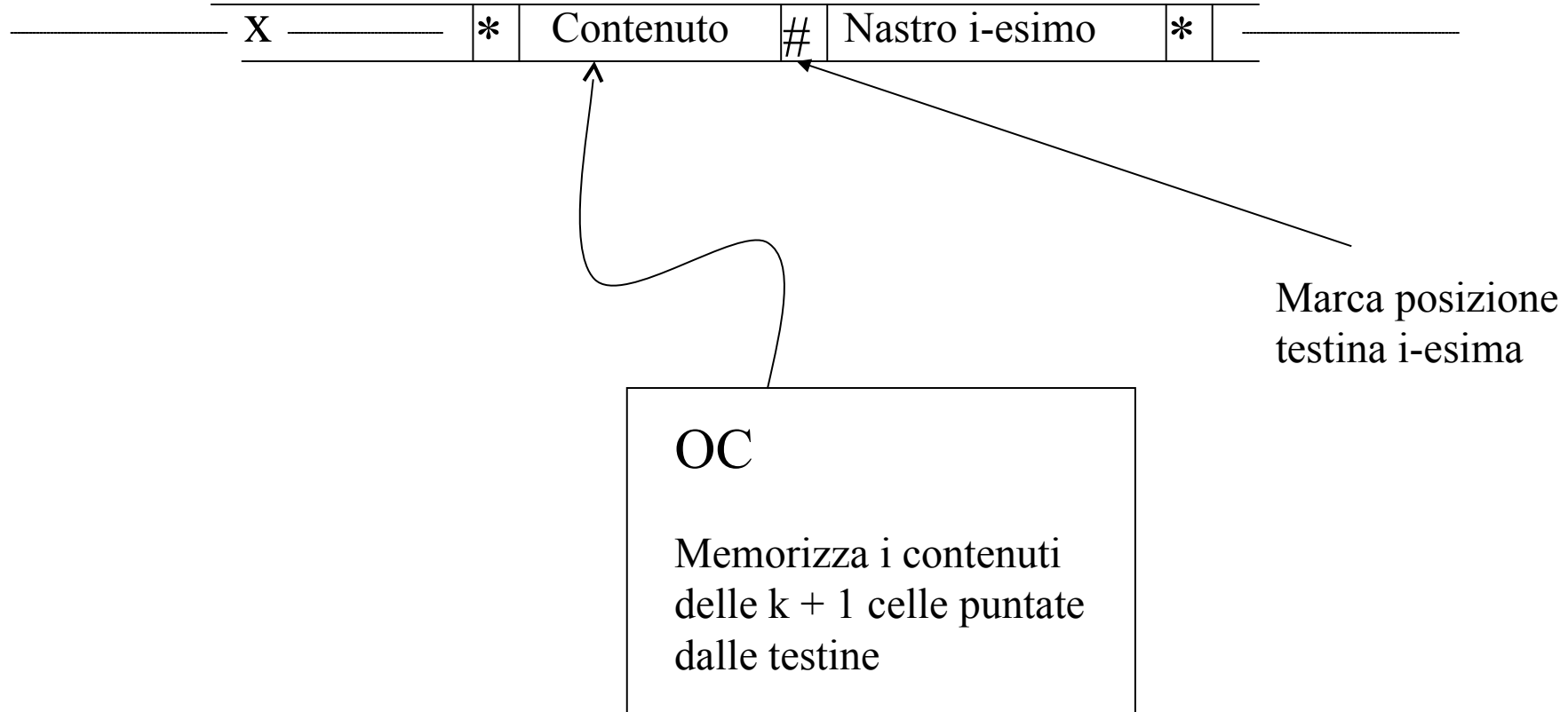
- MT a nastro bidimensionale



- MT a k testine per nastro

-

Le varie versioni di MT sono tutte tra loro equivalenti, rispetto alla capacità riconoscitiva/traduttiva:
ad esempio:



Che relazioni sussistono tra automi vari (MT in particolare) e modelli di calcolo più tradizionali e realistici?

- La MT può simulare una macchina di von Neumann (pur essa “astratta”)
- La differenza fondamentale sta nel meccanismo di accesso alla memoria: sequenziale invece che “diretto”
- La cosa non inficia la potenza della macchina dal punto di vista della capacità computazionale (classe di problemi risolvibili)
- Può esserci invece impatto dal punto di vista della complessità del calcolo
- Esamineremo implicazioni e conseguenze in entrambi i casi

I modelli (operazionali) non deterministici

- Solitamente si tende a pensare ad un algoritmo come una sequenza di operazioni *determinata*: in un certo stato e con certi ingressi non sussistono dubbi sulla “mossa” da eseguire
- Siamo sicuri che ciò sia sempre auspicabile?

Confrontiamo

if $x > y$ **then** $\text{max} := x$ **else** $\text{max} := y$

con

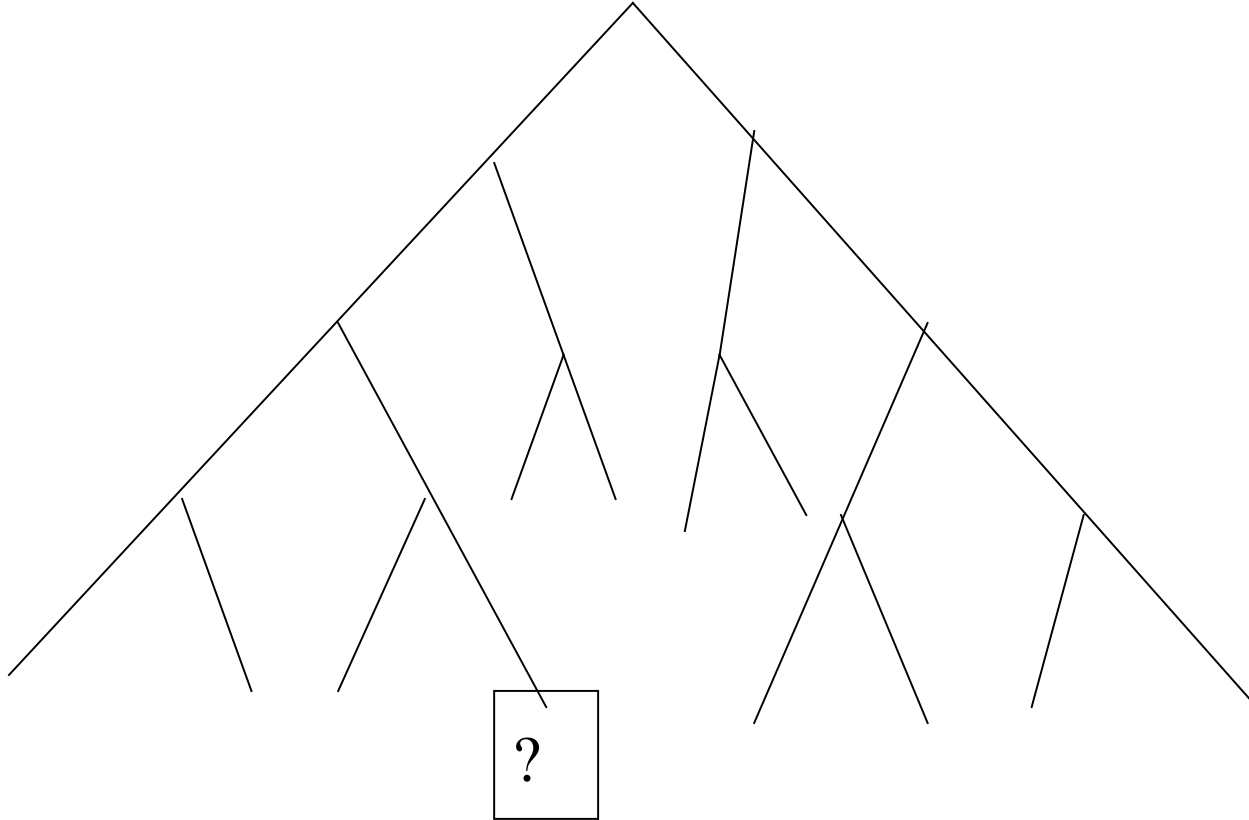
if $x \geq y$ **then** $\text{max} := x$

$y \geq x$ **then** $\text{max} := y$

fi

- E' solo una questione di eleganza?
- Pensiamo al costrutto **case** del Pascal & affini:
perché non un
- **case**
 - $x = y$ **then** S1
 - $z > y + 3$ **then** S2
 - **then** ...
- **endcase**
?

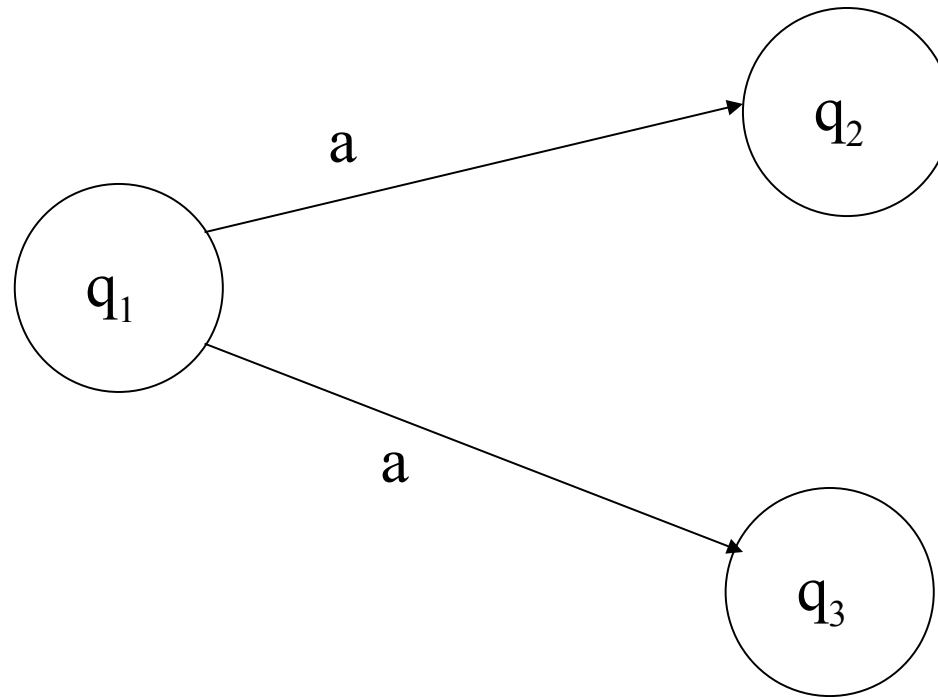
Un'altra forma di nondeterminismo “nascosto”: la ricerca “cieca”



- In realtà i vari algoritmi di ricerca sono una “simulazione” di algoritmi sostanzialmente nondeterministici:
- L’elemento cercato si trova nella radice dell’albero?
- Se sì OK. Altrimenti
 - Cerca nel sottoalbero di sinistra
oppure
 - cerca nel sottoalbero di destra
- scelte o priorità tra le diverse strade sono spesso arbitrarie
- Se poi fossimo in grado di assegnare i due compiti in parallelo a due diverse macchine ---->
- *Nondeterminismo come modello di computazione o almeno di progettazione di calcolo parallelo*
(Ad esempio Ada ed altri linguaggi concorrenti sfruttano il nondeterminismo)

Tra i tanti modelli nondeterministici (ND):
versioni ND dei modelli già noti

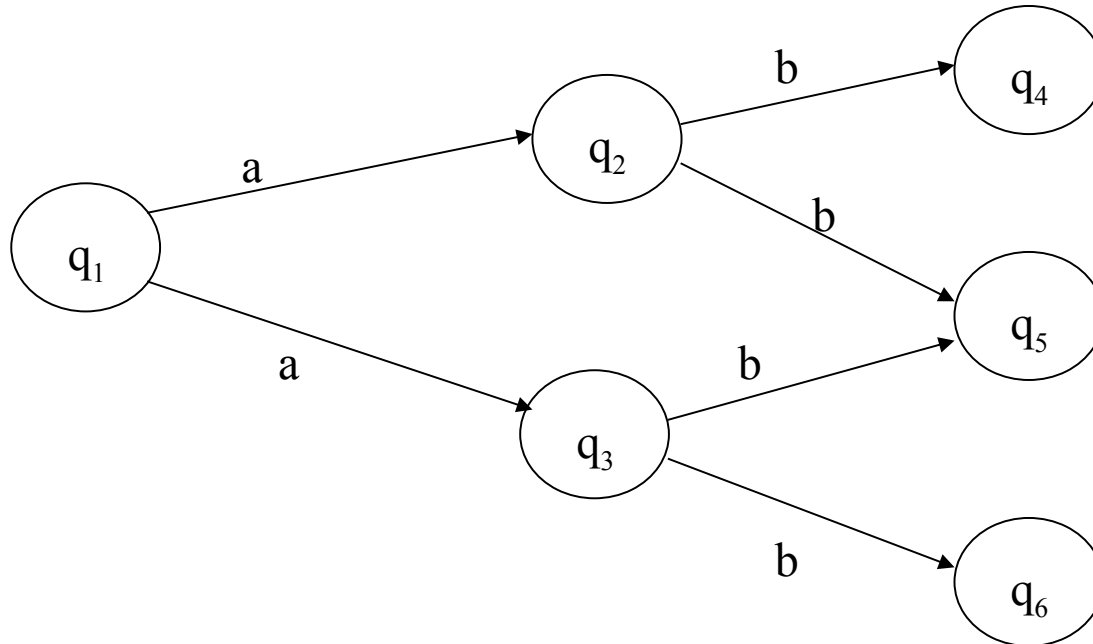
- FSA ND (ne vedremo tra poco la “comodità”)



Formalmente: $\delta(q_1, a) = \{q_2, q_3\}$

$\delta : Q \times I \rightarrow \wp(Q)$

δ^* : formalizzazione della sequenza di mosse



$$\delta(q_1, a) = \{q_2, q_3\}, \delta(q_2, b) = \{q_4, q_5\}, \delta(q_3, b) = \{q_6, q_5\} \quad \delta^*(q_1, ab) = \{q_4, q_5, q_6\}$$

$$\delta^*(q, \varepsilon) = \{q\}$$

$$\delta^*(q, y.i) = \bigcup_{q' \in \delta^*(q, y)} \delta(q', i)$$

Come accetta un FSA ND?

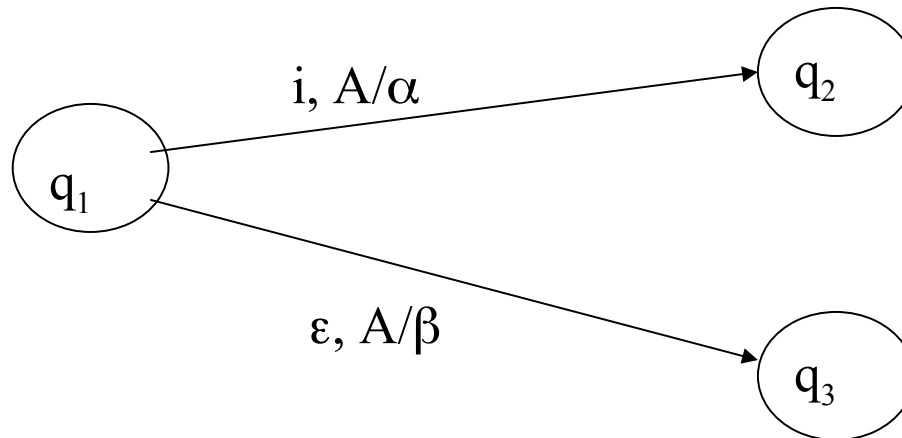
$$x \in L \Leftrightarrow \delta^*(q_0, x) \cap F \neq \emptyset$$

Tra i vari modi di funzionamento dell'automa è sufficiente che uno di essi abbia successo per accettare la stringa di ingresso

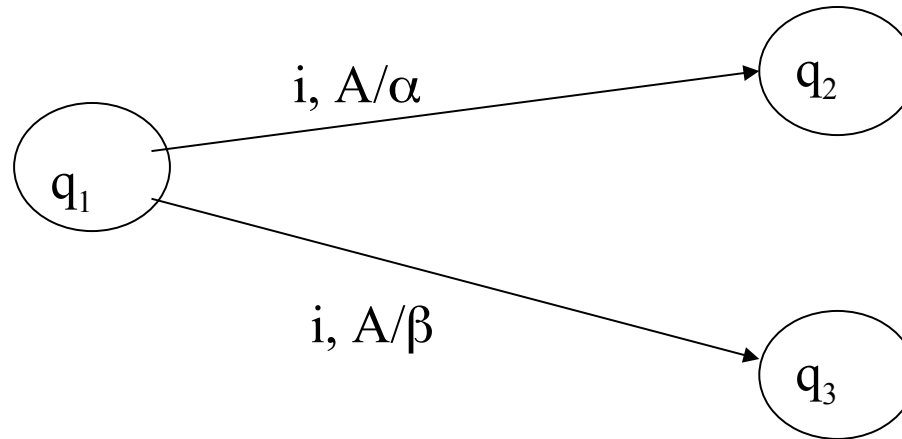
Sono possibili convenzioni diverse, es. $\delta^*(q_0, x) \subseteq F$

Gli AP nondeterministici (APND)

- In realtà essi nascono ND di natura:

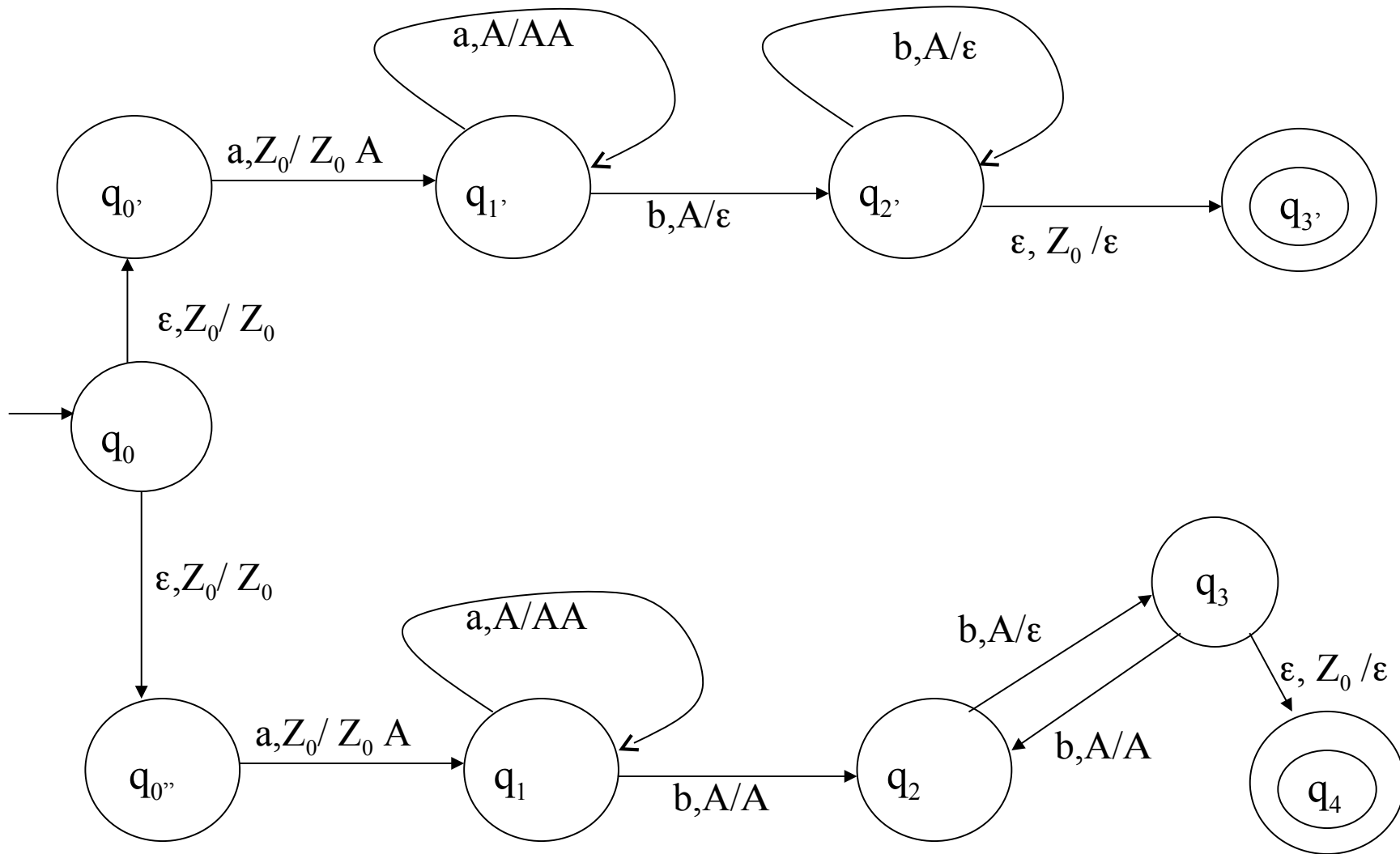


- Tanto vale rimuovere la restrizione del determinismo e generalizzare:



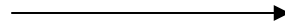
- $\delta : Q \times (I \cup \{\varepsilon\}) \times \Gamma \rightarrow \wp_F(Q \times \Gamma^*)$
- Perché l'indice F? (sta per *Finito*...)
- Al solito l'APND accetta x se *esiste* una sequenza
- $c_0 \vdash^* \langle q, \varepsilon, \gamma \rangle, q \in F$
- \vdash è non più univoca!

Un “banale” esempio



Alcune immediate ma significative conseguenze

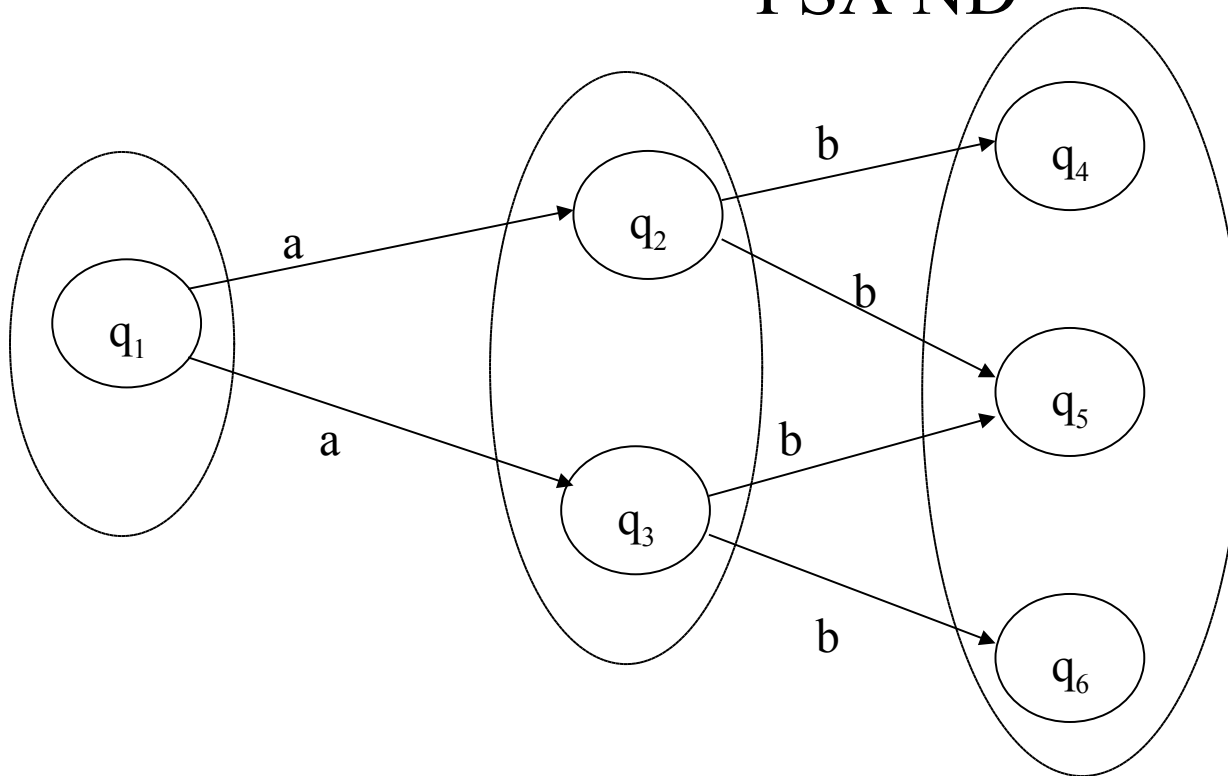
- Gli APND possono riconoscere un linguaggio non riconoscibile dagli AP deterministici ---->
sono più potenti
- La costruzione precedente può essere facilmente generalizzata ottenendo una dimostrazione *costruttiva* (come altre precedenti) di chiusura rispetto all'unione degli APND
-proprietà non sussistente per gli AP deterministici
- La chiusura rispetto all'intersezione invece continua a non sussistere ($\{a^n b^n c^n\} = \{a^n b^n c^*\} \cap \{a^* b^n c^n\}$ non è riconoscibile mediante una pila, neanche in modo ND)
-I due controesempi precedenti $\{a^n b^n c^n\}$ e $\{a^n b^n\} \cup \{a^n b^{2n}\}$ non sono poi così simili tra loro ...



- Se una famiglia di linguaggi è chiusa rispetto all'unione e non rispetto all'intersezione non può essere chiusa rispetto al complemento (perché?)
- Ciò mette in evidenza il profondo cambiamento causato dal nondeterminismo rispetto alla complementazione di un problema - in generale -:
se il modo di funzionamento della macchina è univoco e se la sua computazione giunge al termine, allora:
- sufficiente scambiare la risposta positiva con quella negativa per ottenere la soluzione di un “problema complemento” (ad esempio *presenza* invece di *assenza* di errori in un programma)

- Nel caso degli APND pur essendo possibile, come per gli APD, far sì che una computazione giunga sempre al termine, potrebbero darsi due computazioni
 - $c_0 \vdash^* \langle q_1, \varepsilon, \gamma_1 \rangle$
 - $c_0 \vdash^* \langle q_2, \varepsilon, \gamma_2 \rangle$
 - $q_1 \in F, q_2 \notin F$
- In questo caso x è accettata
- Però se scambio F con $Q-F$, x continua ad essere accettata: nell'ambito del nondeterminismo scambiare il sì con il no non funziona!
- E gli altri tipi di automi?

FSA ND



Partendo da q_1 e leggendo ab l'automa si trova in uno stato che appartiene all'insieme $\{q_4, q_5, q_6\}$

Chiamiamo nuovamente “stato” l'insieme dei possibili stati in cui si può trovare l'automa ND durante il suo funzionamento.

Sistematizzando ...

- Dato un FSA ND ne costruisco *automaticamente* uno equivalente deterministico ---->
- gli automi FSA ND non sono più potenti dei loro fratelli deterministici (diversamente dagli AP) (e allora a che cosa servono?)
- $A_{ND} = (Q_N, I, \delta_N, q_{0N}, F_N)$
- $A_D = (Q_D, I, \delta_D, q_{0D}, F_D)$
 - $Q_D = \wp(Q_N)$
 - $\delta_D(q_D, i) = \bigcup_{q_N \in q_D} \delta_N(q_N, i)$
 - $q_{0D} = \{q_{0N}\}$
 - $F_D = \{Q' \subseteq Q \mid Q' \cap F_N \neq \emptyset\}$

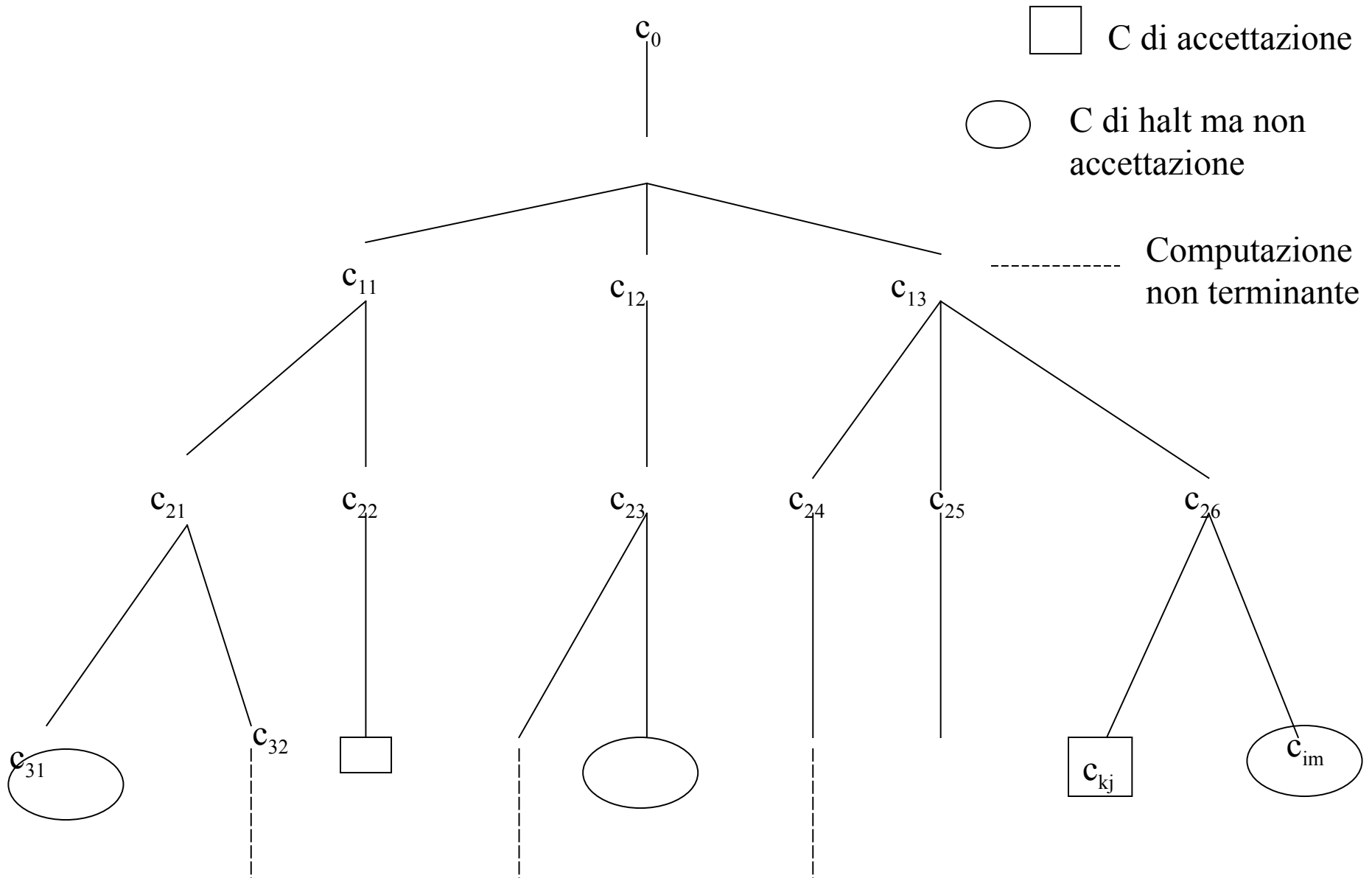
- E' bensì vero che, per ogni automa FS ND ne posso trovare (e *costruire*) uno equivalente deterministico
- Ciò non significa che sia superfluo usare gli FSA ND:
 - Può essere più facile “progettare” un AND e poi ricavarne automaticamente uno deterministico, risparmiandosi la fatica di costruirlo noi stessi deterministico fin da subito (ne vedremo un'applicazione tra breve)
 - Da un AND a 5 stati (ad esempio), ne ricavo, nel caso pessimo, uno con 2^5 stati!
- Resta da esaminare la MT ...

Le MT nondeterministiche

$$\delta, [\eta] : Q \times I \times \Gamma^k \rightarrow \wp(Q \times \Gamma^k \times \{R, L, S\}^{k+1} [\times O \times \{R, S\}])$$

- è necessario l'indice F ?
- configurazioni, transizioni, sequenze di transizioni e accettazione sono definite come al solito
- il nondeterminismo aumenta la potenza delle MT?

Albero delle computazioni



- x è accettata da una MT ND se e solo se esiste una computazione della MND che termina in uno stato di accettazione
- può una MT deterministica stabilire se una sua “sorella” ND accetta x , ossia accettare a sua volta x se e solo se la MND la accetta?
- Si tratta di percorrere o “visitare” l’albero delle computazioni ND per stabilire se esiste in esso un cammino che termina in uno stato di accettazione
- Questo è un (quasi) normale e ben noto problema di visita di alberi, per il quale esistono classici algoritmi di visita
- Il problema è perciò ridotto ad implementare un algoritmo di visita di alberi mediante MT: compito noioso ma sicuramente fattibile ... a meno del “quasi” di cui sopra ...

- Tutto facile se l'albero delle computazioni è finito
- Però potrebbe darsi il caso che alcuni cammini dell'albero siano infiniti (descrivono computazioni che non terminano)
- In tal caso, un algoritmo di visita depth-first (ad esempio, in preordine sinistro) potrebbe “infilarsi in un cammino infinito” senza scoprire che in un altro punto dell'albero ne esiste uno finito che porta all'accettazione.
- Il problema è però facilmente risolvibile adottando ad esempio un algoritmo di visita di tipo breadth-first (che usa una struttura a coda invece di una a pila per accumulare i vari nodi da esaminare: ne parleremo più avanti).

Conclusioni

- Nondeterminismo: utile astrazione per descrivere problemi/algoritmi di ricerca; situazioni in cui non esistono elementi di scelta, o sono tra loro indifferenti; computazioni parallele
- In generale non aumenta la potenza di calcolo, almeno nel caso delle MT (che sono l'automata più potente tra quelli visti finora) però può fornire descrizioni più compatte
- Aumenta la potenza degli automi a pila