Modelli operazionali

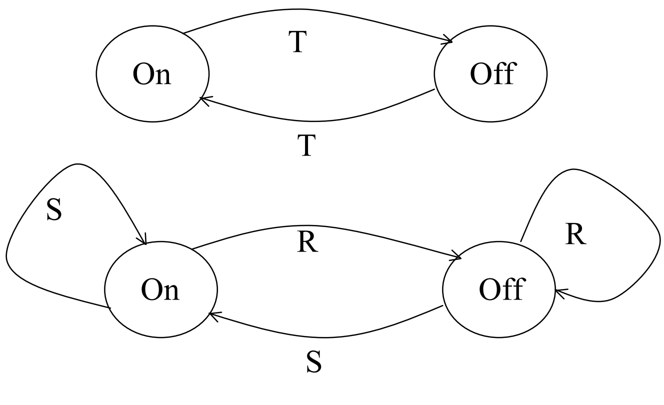
(macchine a stati, sistemi dinamici)

- 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:



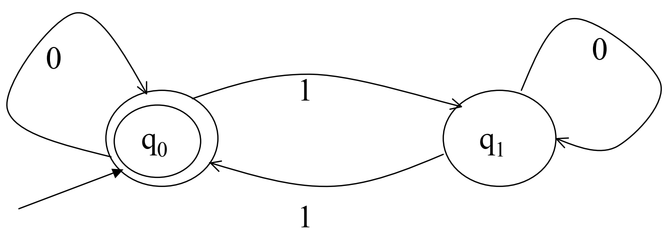
Accensione e spegnimento di luce, ...

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*):
    δ: $Q \times I \rightarrow Q$

L'automa 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*.

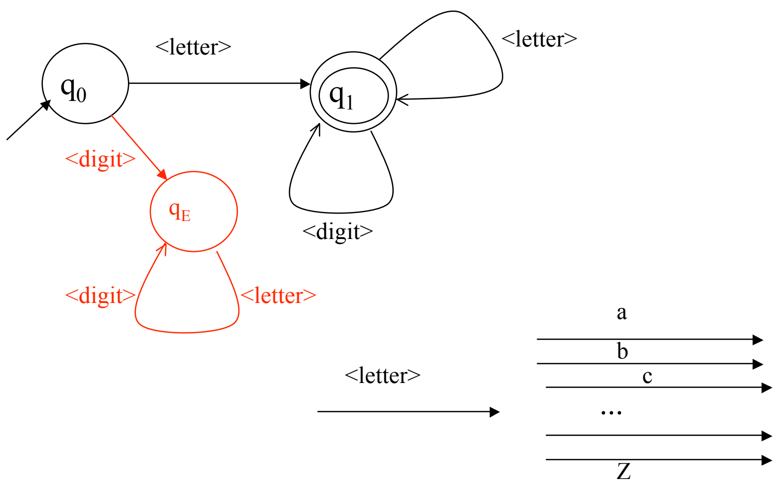


L = {stringhe con un numero pari di "1" e un numero qualsiasi di "0"}

Formalizzazione del riconoscimento di L

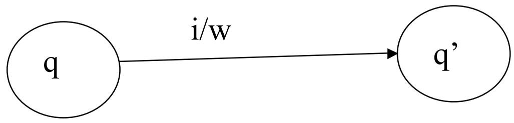
- Sequenza di mosse:
  - $\delta^*: Q \times I^* \rightarrow Q$
    δ^* definita induttivamente a partire da δ
    $\delta^* (q,\varepsilon) = q$
    $\delta^* (q,y \cdot i) = \delta(\delta^* (q,y), i)$ $\qquad (y \in I^*, i \in I)$
- Stato iniziale: $q_0 \in Q$
- Stat*i* final*i* o di accettazione: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

Riconoscimento di identificatori Pascal



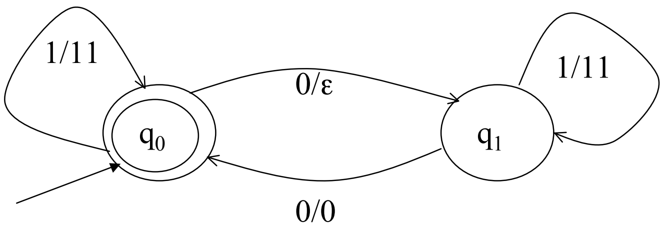
L'automa come traduttore di linguaggi
$y = \tau(x)$

Transizione con uscita:



$i \in I$
$w \in O^*$

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

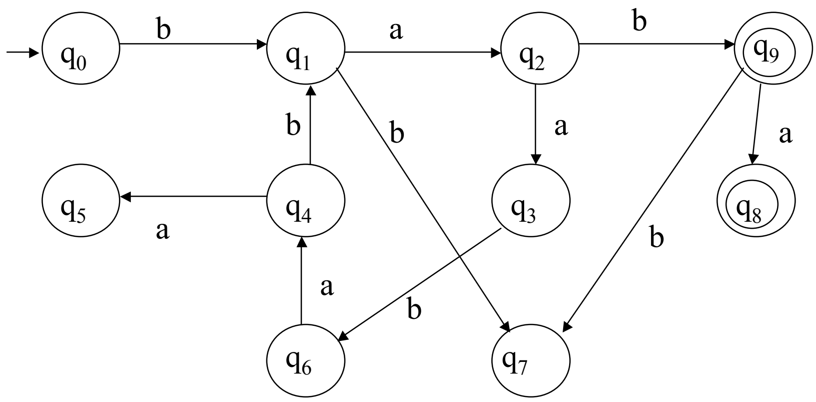


Formalizzazione degli automi traduttori

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

Analisi del modello a stati finiti
(per la sintesi si rimanda ad altri corsi - e.g. calcolatori)

- Modello molto semplice ed intuitivo, applicato in molteplici settori, anche fuori dall'informatica
- Si pagherà un prezzo per tale semplicità?
- …
- Una prima proprietà fondamentale: il *comportamento ciclico* degli automi a stati finiti



C'è un ciclo q_1 ----aabab---> q_1

Se un ciclo è percorribile una volta, esso è anche percorribile 2, 3, …, n, … 0 volte =========>

Più formalmente:

- Se $x \in L$ e $|x| > |Q|$ esistono un $q \in Q$ e un $w \in I^+$ tali che:
  $x = ywz$ $(y,z \in I^*)$
  $\delta^* (q,w) = q$

==============>
$yw^nz \in L, \forall\ n \geq 0$

Pumping Lemma

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

- $L = \emptyset$? $\exists\ x \in L \leftrightarrow \exists\ y \in L, |y| < |Q|$: Basta "eliminare tutti i cicli" dal funzionamento dell'automa che riconosce x
- $|L| = \infty$? Ragionamento simile
- …
- Si noti che, in generale, saper rispondere alla domanda "$x \in L$?" per un generico x *non* implica saper rispondere alle altre domande!!

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

- Più intuitivamente: per "contare" n qualsiasi occorre una memoria infinita!
- Rigorosamente parlando ogni calcolatore è un FSA, però …astrazione sbagliata! Importanza del "concetto astratto di infinito"!
- 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"

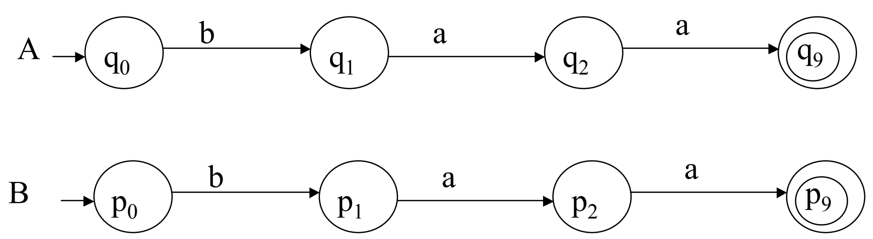
Le proprietà di *chiusura* dei FSA

- 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)

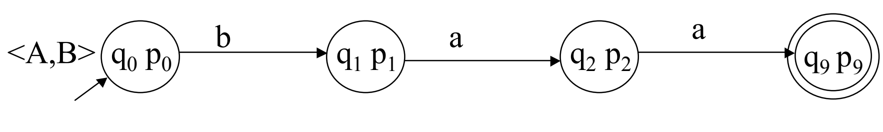
Nel caso dei linguaggi:

- $\mathsf{L} = \{L_i\}$: *famiglia* di linguaggi
- L è chiusa rispetto a OP se e solo se per ogni $L_1, L_2 \in \mathsf{L}$, L_1 OP $L_2 \in \mathsf{L}$.
- R : linguaggi *regolari*, riconosciuti da FSA
- R chiusa rispetto alle operazioni insiemistiche, alla concatenazione, la "*", … e praticamente "tutte" le altre.

Intersezione



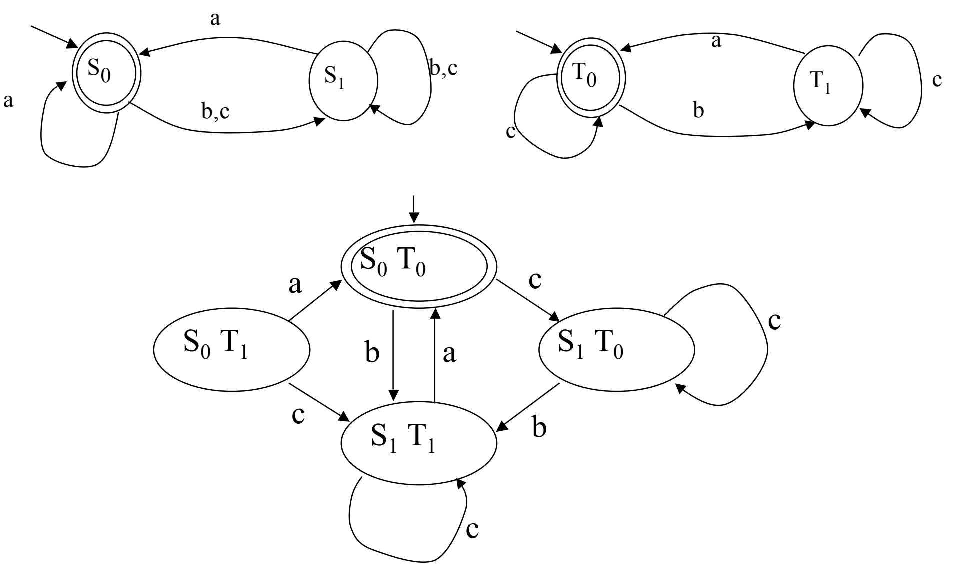
Posso simulare il "funzionamento parallelo" di A e B semplicemente "accoppiandoli":



Formalmente:

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

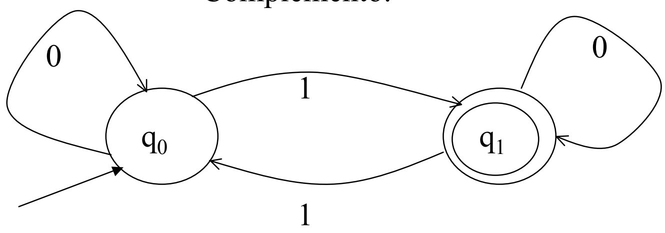
Intersezione: esempio



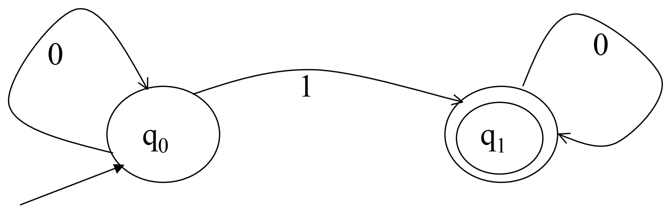
Unione

- Costruzione simile …
  oppure ...

Complemento:

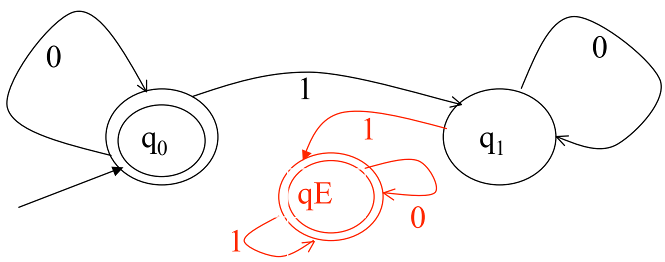


Idea: F^ = Q -F: Sì però ….



Se mi limito a scambiare F con Q - F …

Il problema nasce dal fatto che δ è parziale:



23

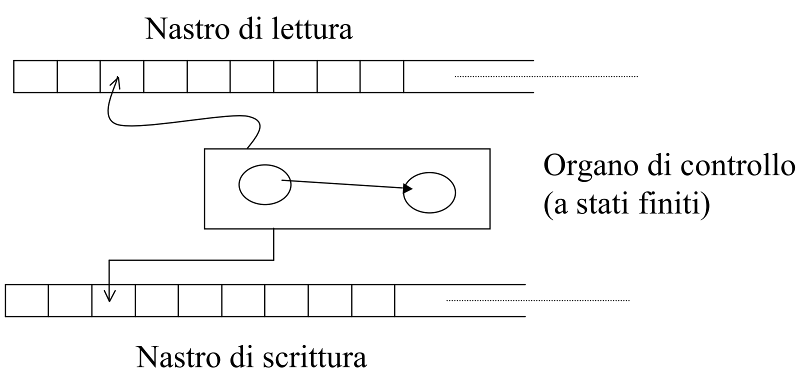
Filosofia generale del complemento

- 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!!

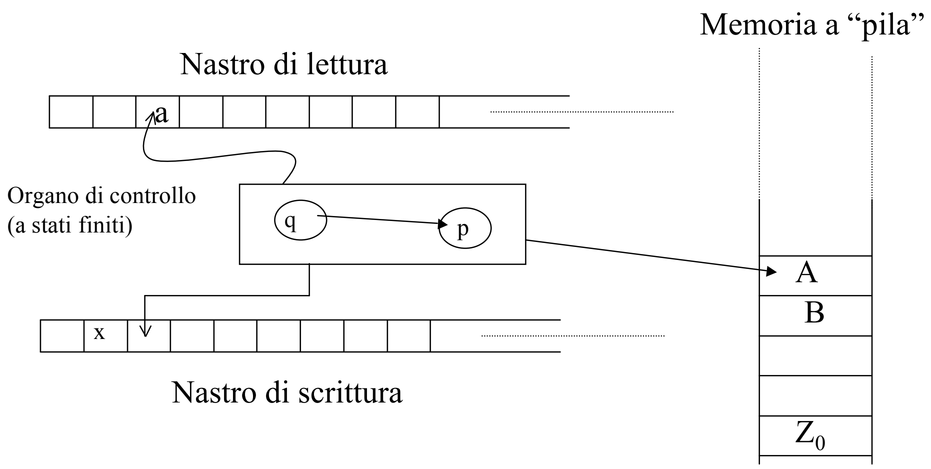
24

Aumentiamo la potenza dei FSA aumentandone la memoria

- Una visione più “meccanica” del FSA:



- Ora “arricchiamolo” un po’:

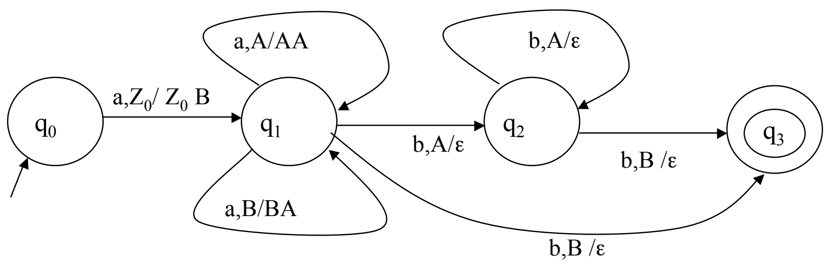


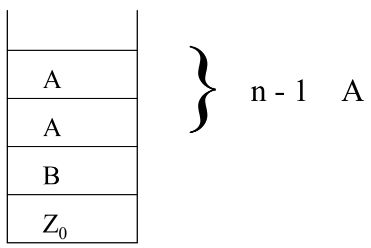
La mossa dell’automa a pila:

- 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)*

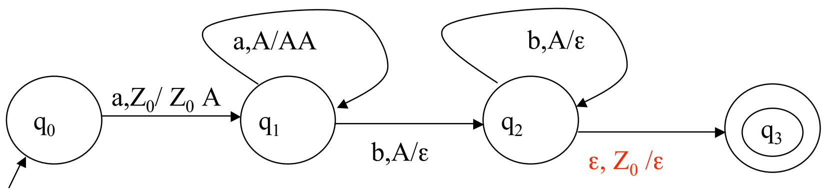
- La stringa di ingresso x viene riconosciuta (accettata) se
  - L'automa la scandisce completamente (la testina di lettura giunge alla fine di x)
  - Giunto alla fine di x esso si trova in uno stato di accettazione (come il FSA)
- Se l'automa è anche traduttore
  $\tau(x)$ è la stringa che si trova 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 $\{a^n b^n \mid n > 0\}$



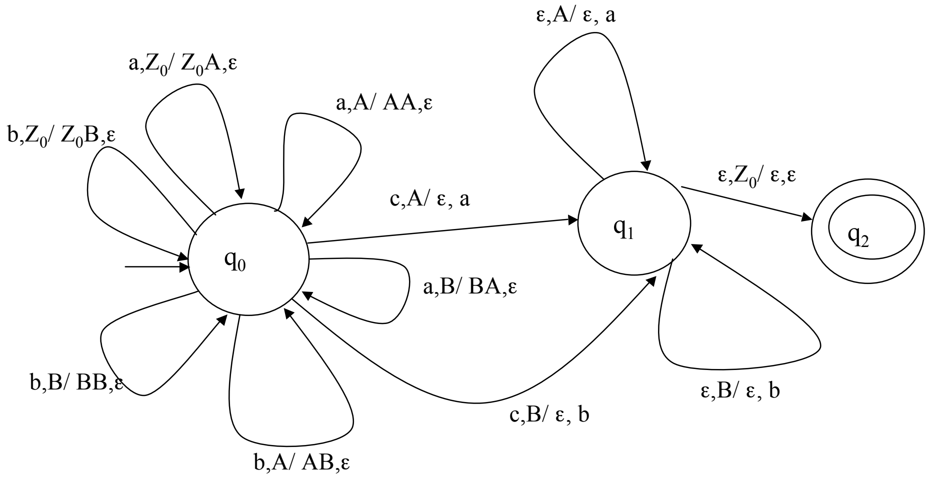


Oppure:



ε- mossa!

Un (classico) automa-traduttore a pila



Cosa traduce? $\tau(wc)\ [w \in \{a,b\}^+] = w^R$ (stringa invertita)

Formalizziamo un po' ...

•Automa [traduttore] a Pila: <$Q, I, \Gamma, \delta, q_0, Z_0$, F [O, η]>

•Q, I, q_0, F [O] come FSA [T]

• Γ alfabeto di pila (per comodità disgiunto dagli altri)

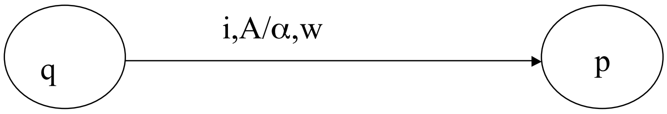
•Z_0 : simbolo iniziale di pila

• δ: $Q \times (I \cup \{\varepsilon\}) \times \Gamma \rightarrow Q \times \Gamma^*$ δ : parziale!

• η: $Q \times (I \cup \{\varepsilon\}) \times \Gamma \rightarrow O^*$ (η definita dove δ è definita)

Notazione grafica:

$\delta(q,i,A) = <p,\alpha>$
$\eta(q,i,A) = w$



- Configurazione (concetto generale di stato):
  c = <q, x, γ, [z]>:
  - 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:
  $c = \langle q, x, \gamma, [z] \rangle$ |-- $c' = \langle q', x', \gamma', [z'] \rangle$
  - $\gamma = \beta A$
  - Caso 1: $x = i \cdot y$ e $\delta(q,i, A) = \langle q', \alpha \rangle$ (è definita)
    $[\eta(q,i, A) = w]$
  - $x' = y$
  - $\gamma' = \beta\alpha$
  - $[z' = z \cdot w]$
  - Caso 2: $\delta(q,\varepsilon, A) = \langle q', \alpha \rangle$ (è definita)
    $[\eta(q, \varepsilon, A) = w]$
  - $x' = x$
  - $\gamma' = \beta\alpha$
  - $[z' = z \cdot w]$
- NB: $\forall q,A,\ \delta(q,\varepsilon, A) \neq \perp \Rightarrow \delta(q, i, A) = \perp\ \forall\ i$.
- Altrimenti … nondeterminismo!

- Accettazione [e traduzione] di una stringa
- |-*- : chiusura transitiva e riflessiva di |--
- $x \in L\ [z = \tau(x)] \leftrightarrow$
- $c_0 = \langle q_0, x, Z_0, [\varepsilon] \rangle$ |-*- $c_F = \langle q, \varepsilon, \gamma, [z] \rangle$, $q \in F$

Occhio alle ε-mosse, soprattutto a fine stringa!!

35

L'automa 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

36

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'automa a pila può essere dimostrata formalmente mediante un'estensione del pumping lemma.
- $\{a^n b^n \mid n > 0\}$ riconoscibile da un automa a pila;
  $\{a^n b^{2n} \mid n > 0\}$ riconoscibile da un automa a pila
- Però $\{a^n b^n \mid n > 0\} \cup \{a^n b^{2n} \mid n > 0\}$ …
  - 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 ….

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 ---> non si giunge mai in fondo alla stringa ----> la stringa non è accettata, ma non è accettata neanche dall'automa con F^ = Q-F.
- Esiste però una costruzione che ad ogni automa associa un automa equivalente loop-free
- 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??

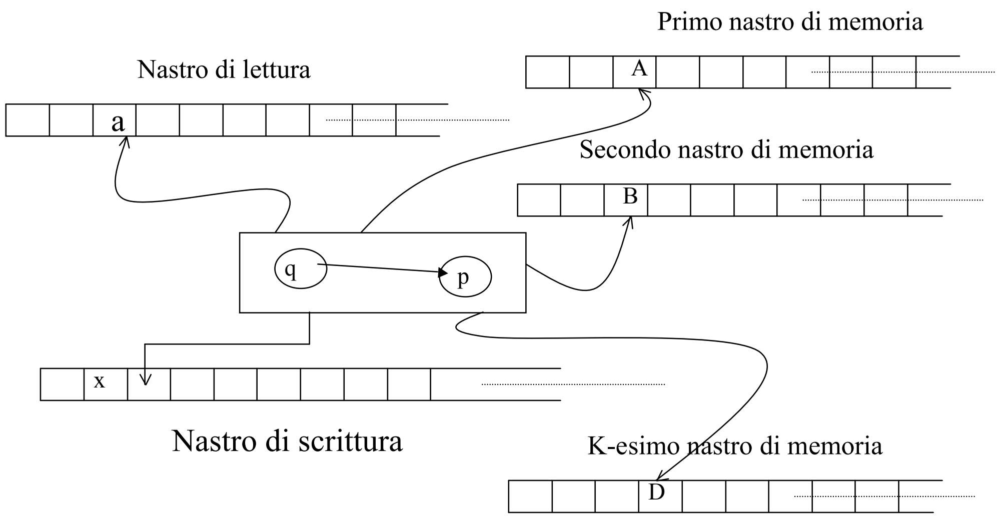
- $<q_1, \varepsilon, \gamma_1> \mid\!-- <q_2, \varepsilon, \gamma_2> \mid\!-- <q_3, \varepsilon, \gamma_3> \mid\!-- \dots$
  $q_1 \in F$, $q_2 \notin F$, …. ?
- Occorre "obbligare" l'automa a decidere l'accettazione solo alla fine di una sequenza (necessariamete finita) di ε-mosse.
- Anche questo è 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;
  - gli AP hanno capacità di conteggio illimitato che gli FSA non hanno
- Anche gli AP/TP hanno i loro limiti …
- … un nuovo e "ultimo" (per noi) automa:
  La *Macchina di Turing* (MT)
- 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.

41

MT a k-nastri



42

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 "tecnicalità matematiche" i nastri sono rappresentati da sequenze *infinite* di celle [0,1,2, …] invece che da stringhe finite. Però esiste un simbolo speciale "blank" (" ", o b "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, pure "simili" alle altre testine

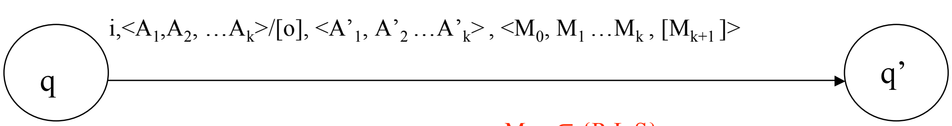
- La mossa della macchina di Turing:
- Lettura:
  - carattere in corrispondenza della testina del nastro di ingresso
  - k caratteri in corrispondenza delle testine dei nastri di memoria
  - stato dell'organo di controllo
- Azione conseguente:
  - cambiamento di stato: q ----> q'
  - riscrittura di un carattere al posto di quello letto su ogni nastro di memoria: A_i ----> A_i', 1 <= i <= 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 "è 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:



$M_{0..k} \in \{R,L,S\}$

$M_{k+1} \in \{R,S\}$

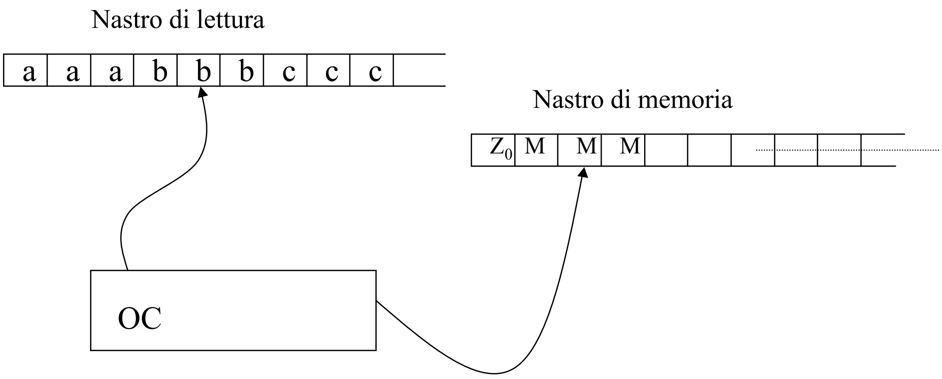
Perché non si perde generalità usando O invece che O^* in uscita?

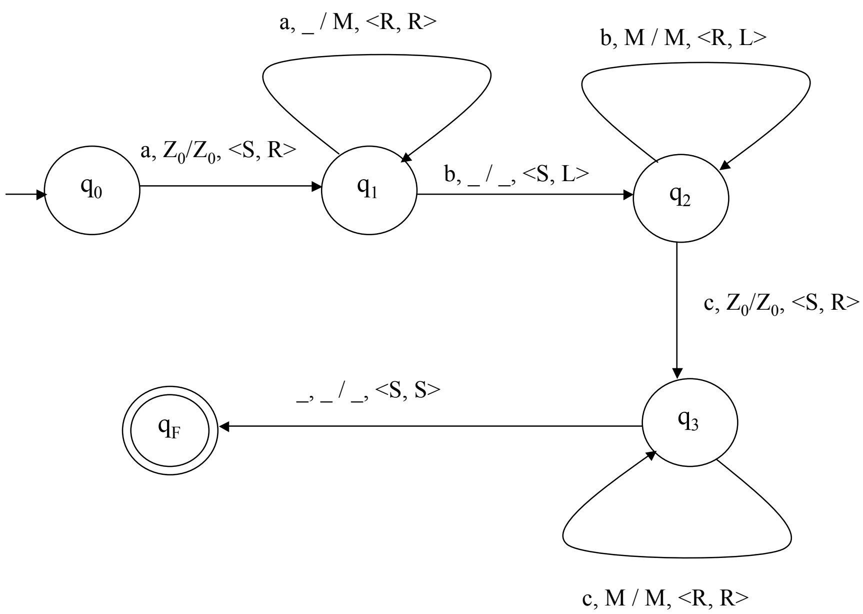
- Configurazione iniziale:
  - Z_0 seguito da tutti blank nei nastri di memoria
  - [nastro di uscita tutto blank]
  - Testine nelle posizioni 0-esime di ogni nastro
  - Stato iniziale dell'organo di controllo q_0
  - Stringa di ingresso x a partire dalla 0-esima cella del nastro corrispondente, seguita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

- Configurazione finale:
  - Stati di accettazione $F \subseteq Q$
  - Per comodità, convenzione:
    $<\delta,[\eta]> (q, \ldots) = \perp \ \forall\, q \in F$:
  - La macchina si ferma quando $<\delta,[\eta]> (q, \ldots) = \perp$
  - 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 $<\delta,[\eta]> (q, \ldots) = \perp$
    - 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'e` 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 $\{a^n b^n c^n \mid n > 0\}$

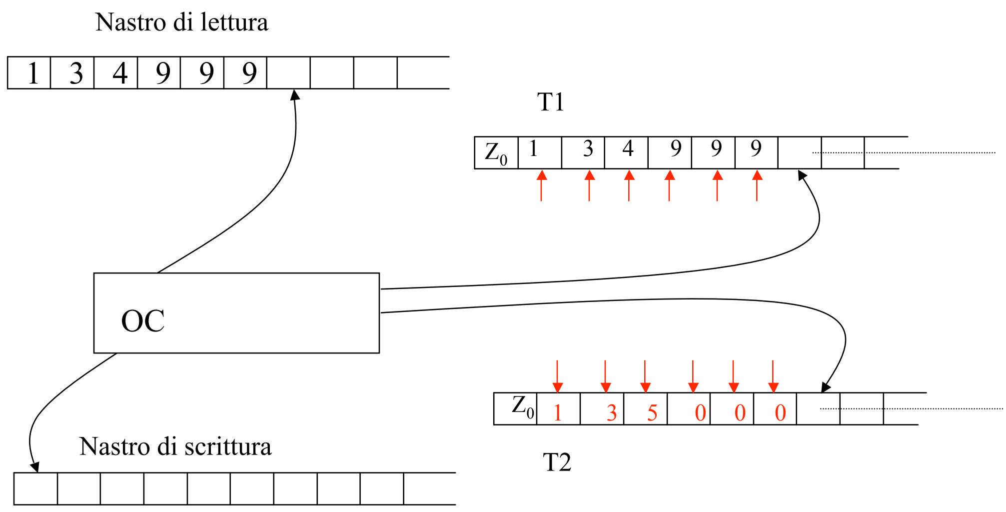


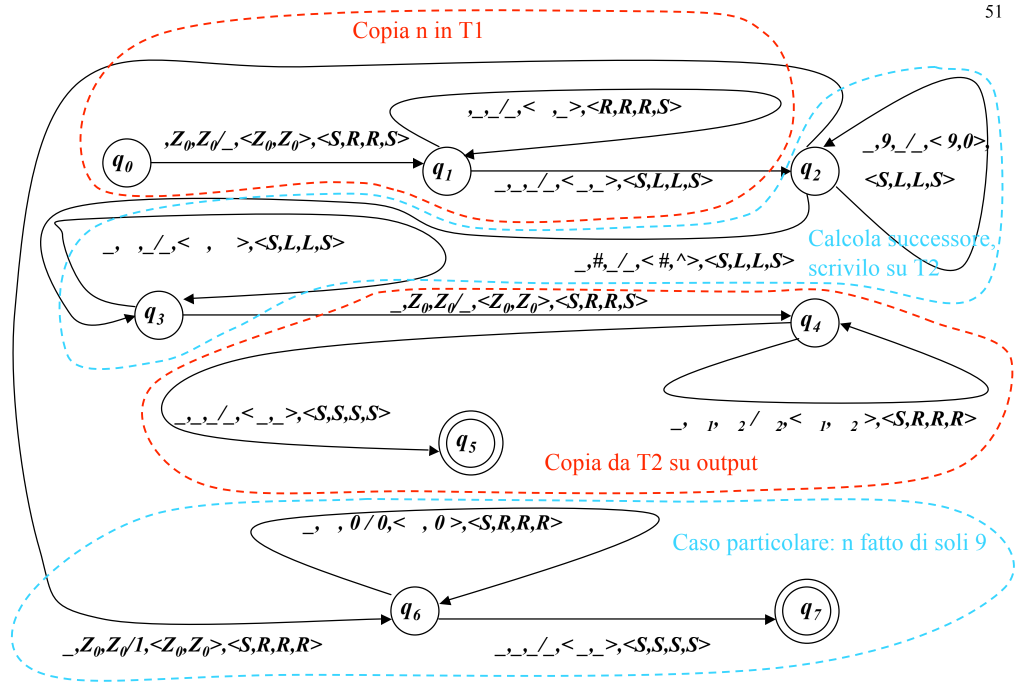


Calcolo del successore di un numero codificato in cifre decimali

- 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
  ^ ***: il successore della cifra denotata da # (nella stessa transizione)***





Proprietà di chiusura delle MT

- ∩ : OK (una MT può facilmente simularne due, sia “in serie” che “in parallelo”)
- ∪ : 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.

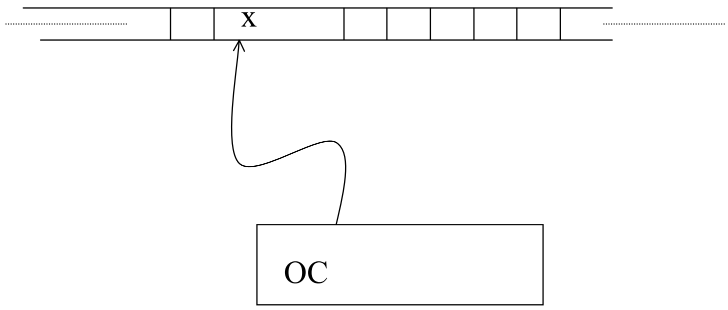
==========>

Evidentemente il problema sta nelle *computazioni che non terminano*

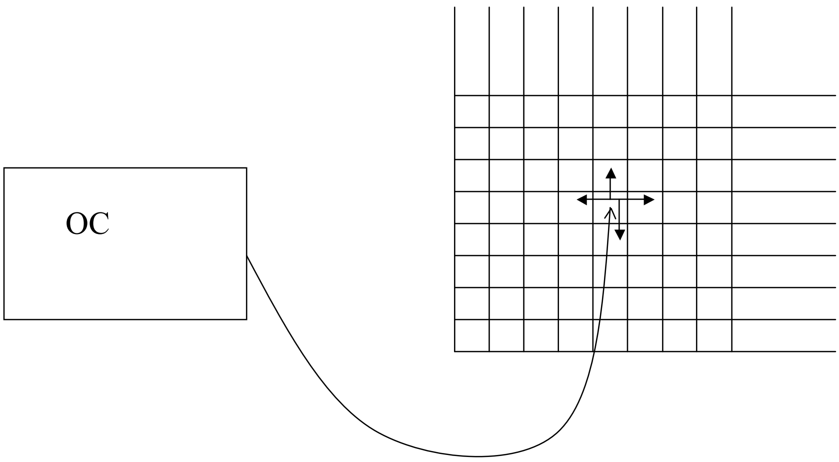
Modelli equivalenti di MT

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

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

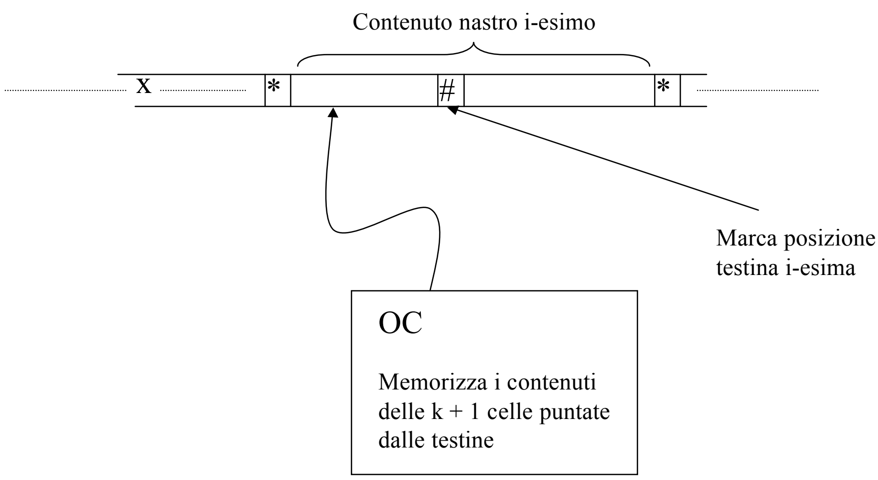


- MT a nastro bidimensionale



- 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