Automi a stati finiti

Fondamenti di Informatica I
Corso di laurea in Ingegneria Informatica e Automatica
Sapienza Università di Roma

A.A. 2018-19

Domenico Lembo, Paolo Liberatore,
Alberto Marchetti Spaccamela, Marco Schaerf

Automi a stati finiti

sistema per dire se una stringa è fatta in un certo modo

come per le espressioni regolari, dividono l'insieme delle stringhe su un certo alfabeto in due sottoinsiemi:

- stringhe riconosciute
- stringhe non riconosciute

Espressioni Regolari vs. Automi

- espressioni regolari

  si scrive la forma delle stringhe da cercare

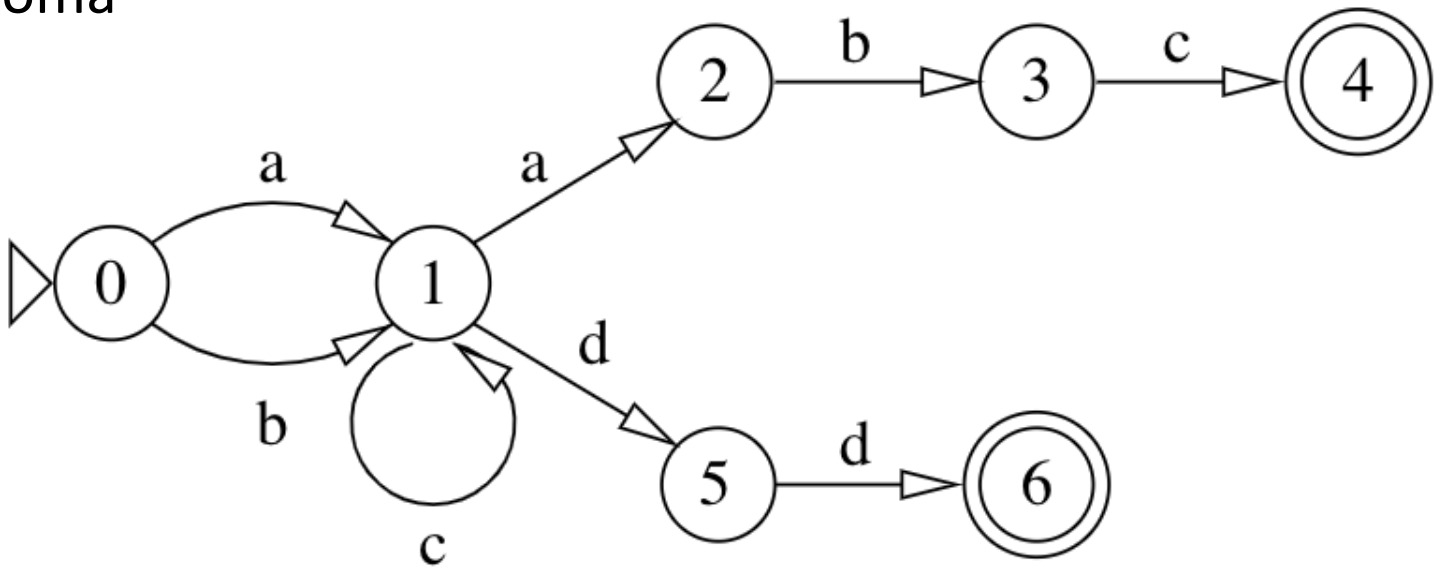
- automi

  meccanismo che opera passo per passo sui caratteri

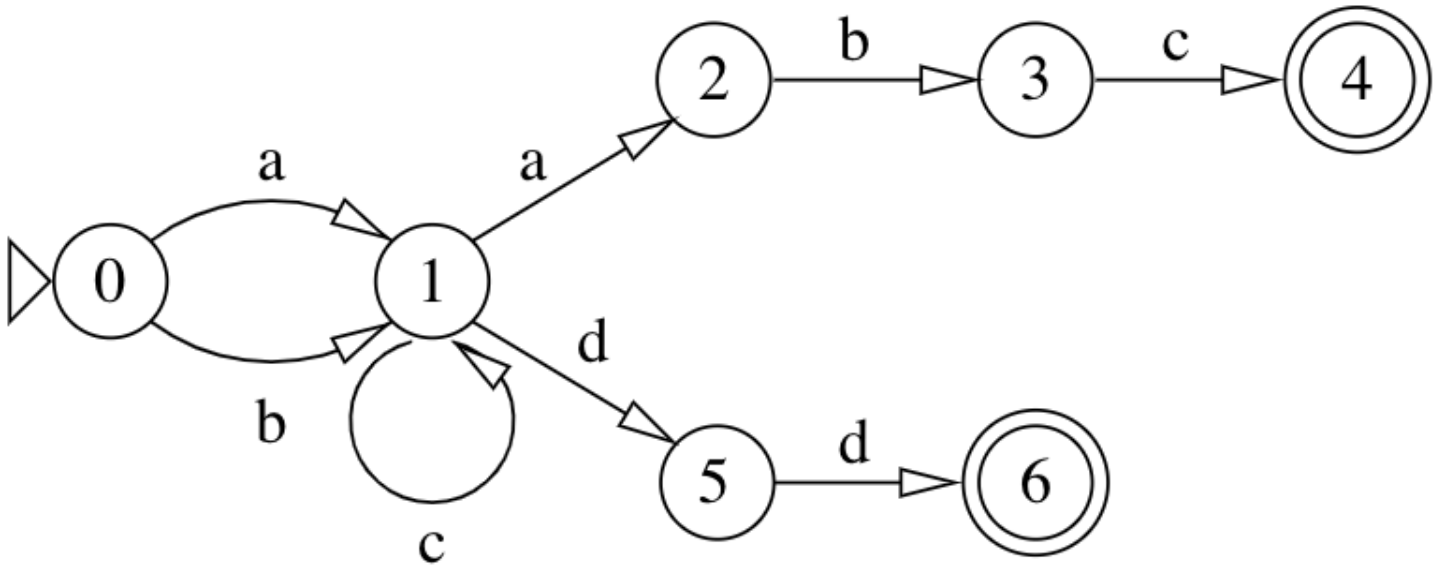
espressione e automa: esempio

[ab]c*(abc|dd) un carattere fra a e b, un numero qualsiasi di c, poi abc oppure dd

automa



Automa



cerchietti = stati in cui può trovarsi l'automa

- inizio: stato `0`
- primo carattere `a` o `b` si va in `1`
- con carattere `c`: si resta in `1`
- con `a` si va in `2`

…

Riconoscimento e definizione

Se alla fine della stringa ci si trova in uno stato con doppio cerchio la stringa è accettata

automi = espressioni regolari

(automa accetta se e solo se espressione collima)

Definizione di automa:

- stati (i pallini)
- collegati fra loro da archi (le frecce)
- con etichette (i caratteri sopra le frecce)

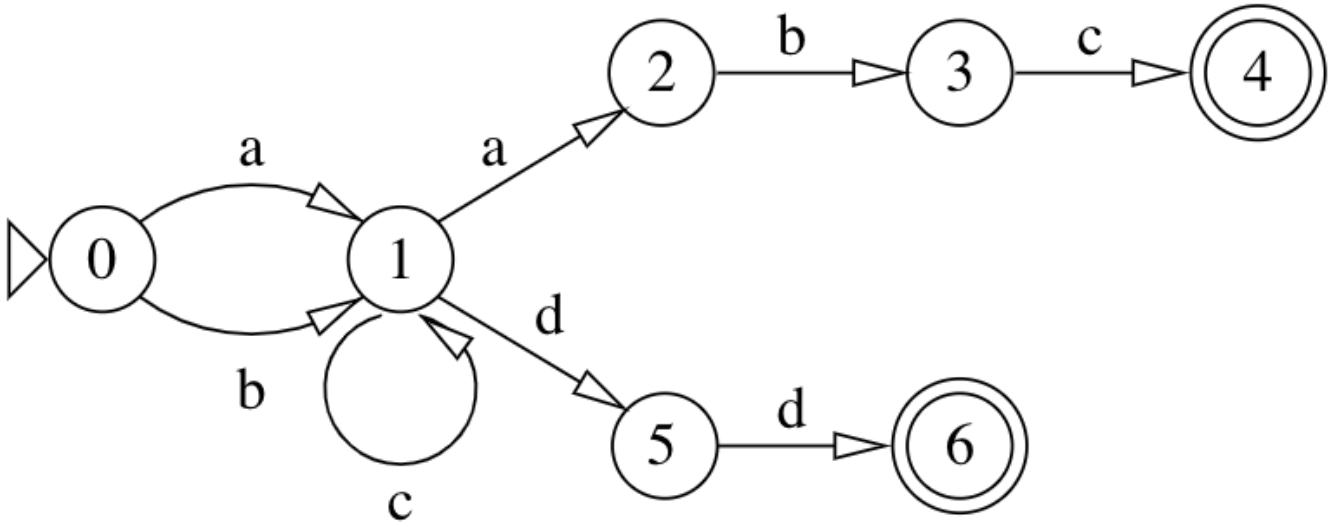
funzionamento…

funzionamento degli automi

- inizio: *stato iniziale*
- la stringa viene fornita un carattere per volta
- se l'automa è nello stato `i` e riceve il carattere `c` segue la freccia da `i` che ha il carattere `c`
- accettazione/rifiuto:
  - freccia non esiste: stringa rifiutata
  - se alla fine in stato finale: stringa accettata

funzionamento: esempio

Automa di prima con stringa ‘accabc’



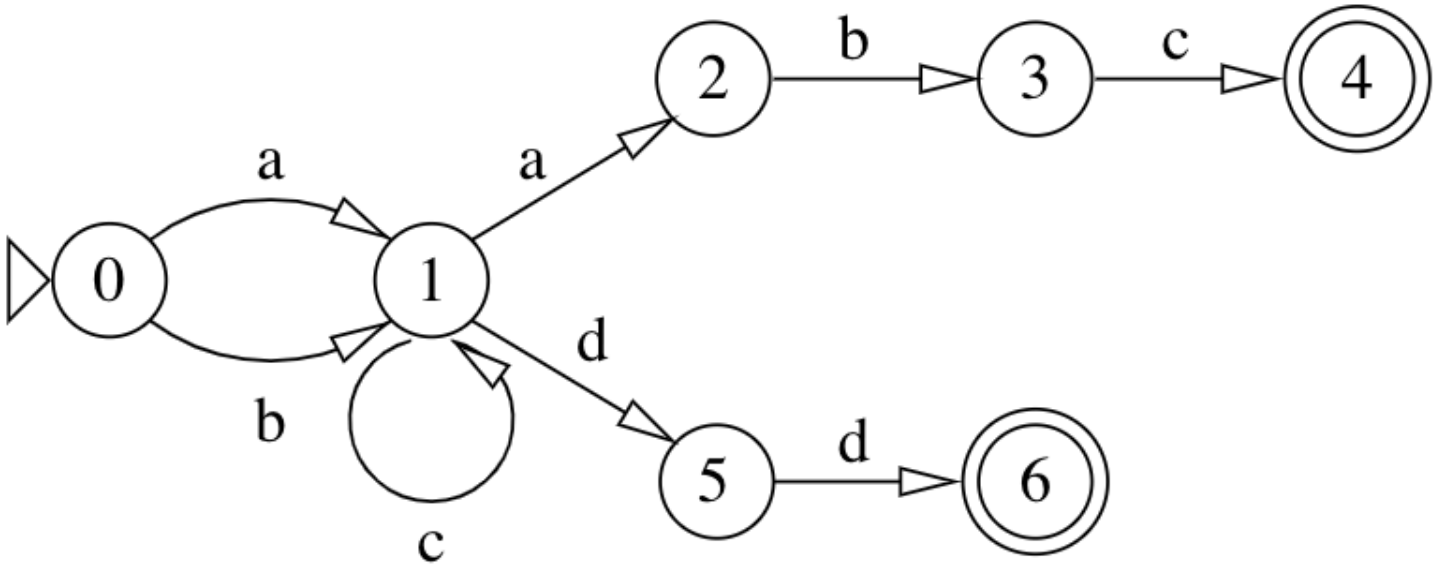
Stringa a c c a b c

Fine stringa -> stato finale

Stringa accettata

esempio con stringa sbagliata

Stringa ‘aacb’

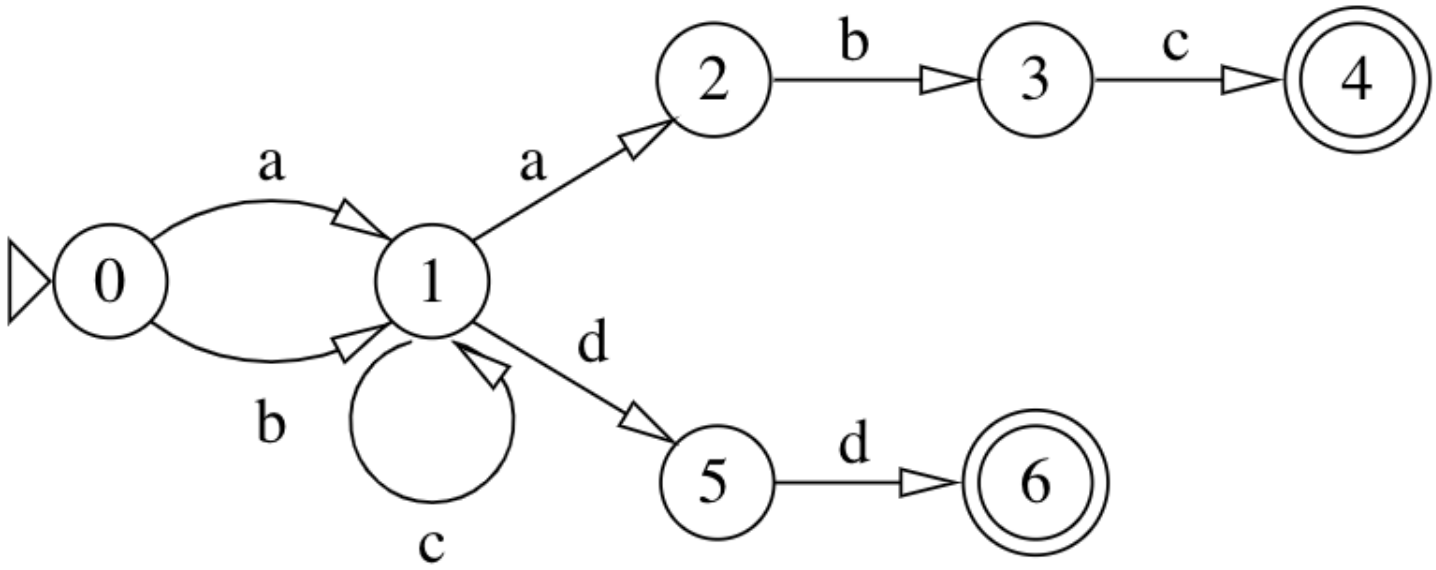


stringa: a a c b

no uscita da stato → stringa rifiutata

esempio con stringa troppo corta

Stringa ‘bcab’



Stringa: b c a b

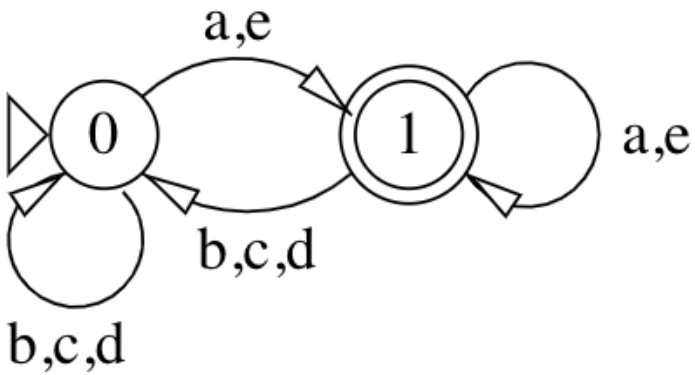
Ultimo stato non doppio → stringa rifiutata

stringhe rifiutate

due possibili motivi:

1. si arriva a uno stato che non ha archi uscenti etichettati con il prossimo carattere
2. si termina su uno stato non finale

Esempio:



Accetta le stringhe che:

- sono fatte di caratteri a,b,c,d,e
- finiscono con una vocale fra queste

uguale all'espressione regolare `^[abcde]*[ae]$`

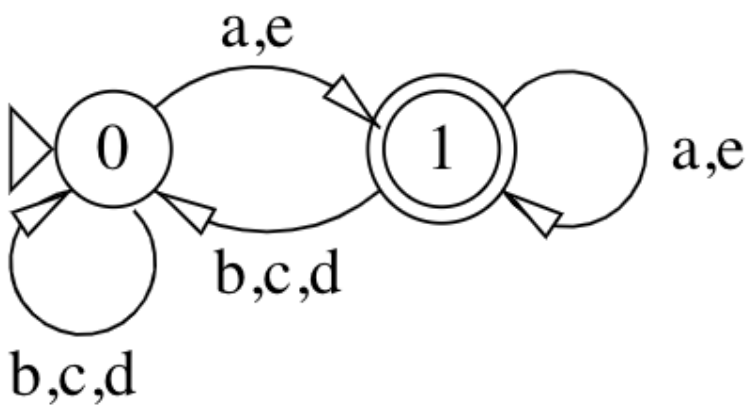
arco con più lettere: seguire per una qualsiasi di queste

cappi = archi che portano allo stesso stato

progettazione degli automi

- capire quali stati servono
- (es: ultima lettera vocale / consonante)
- Inserire gli archi

Stato finale non vuol dire che non si può andare oltre



qui: da 1 (finale) si va in 0

stato finale = se si finisce lì, la stringa è accettata

automi: note

- frecce con più caratteri: va seguita se ne arriva uno qualunque fra quelli indicati
- una freccia può portare al nodo stesso (cappio)
- si può andare oltre lo stato finale

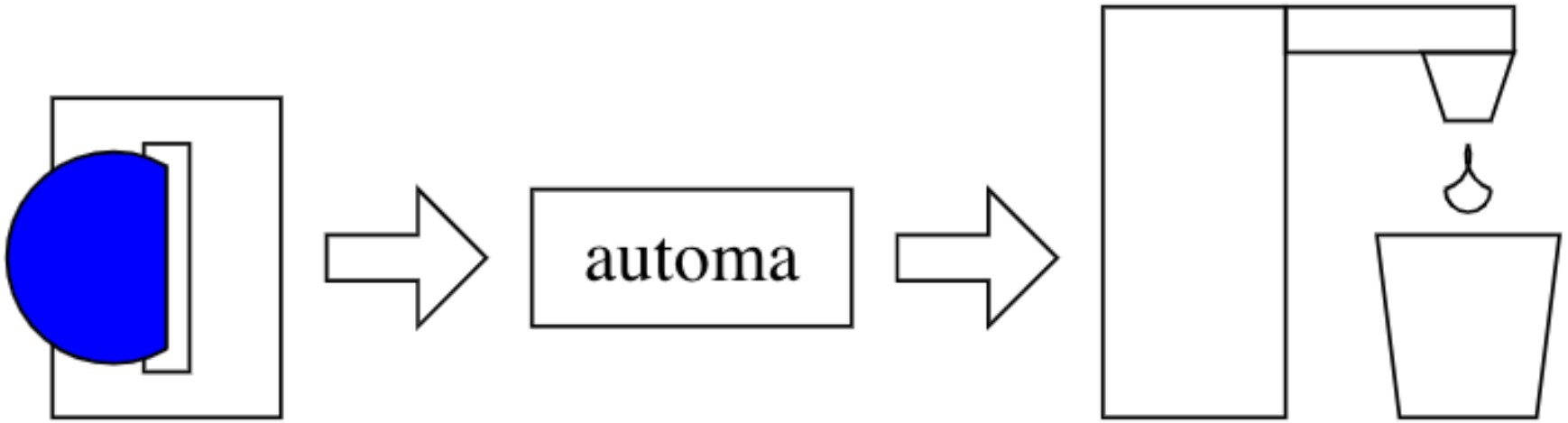
macchinetta del caffè

ingresso: monete da 5 e 10 centesimi

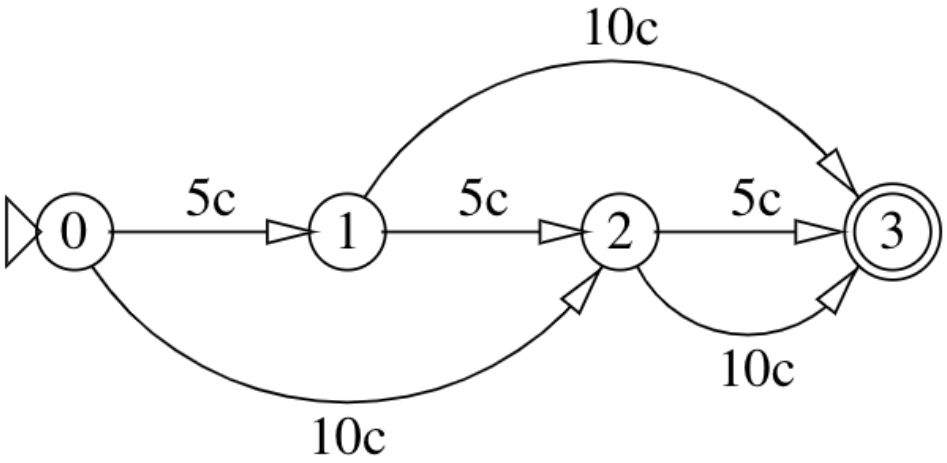
eroga con **almeno** 15 centesimi

(niente resto)

omesso: ricomincia dall'inizio dopo l'erogazione



automa



- Stati: indicano i centesimi inseriti
- Stati finali: abbastanza centesimi per il caffè

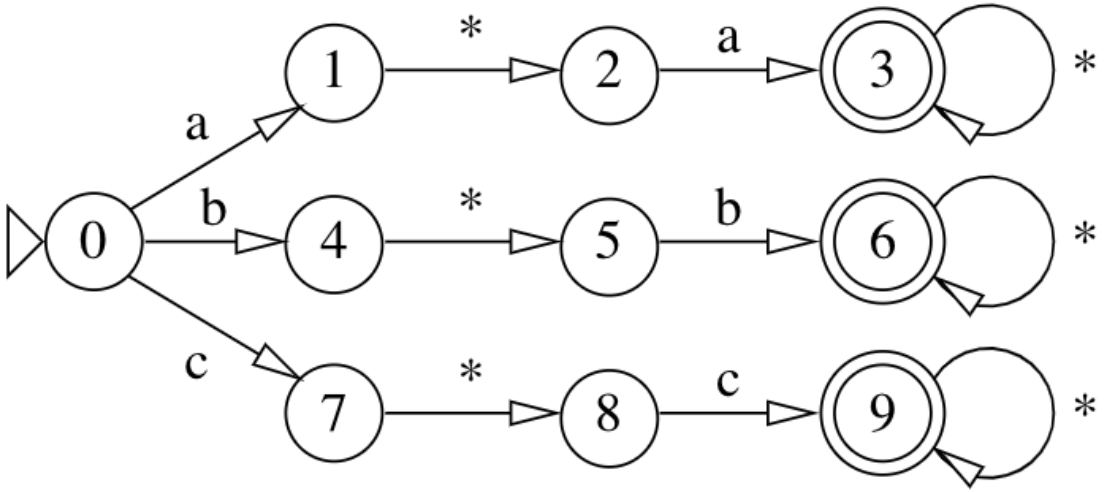
Per ora niente resto, vedremo più avanti

stato = memoria

- esempio:
  - automa che accetta le stringhe che hanno il terzo carattere uguale al primo
- serve "memorizzare" il primo carattere
- stato-memoria = stati diversi a seconda del primo carattere

- realizzare l'automa che accetta qualsiasi stringa in cui il primo carattere è uguale al terzo
- caratteri a, b, c

terzo uguale al primo: soluzione



* = qualsiasi carattere, in questo caso: uguale ad a,b,c

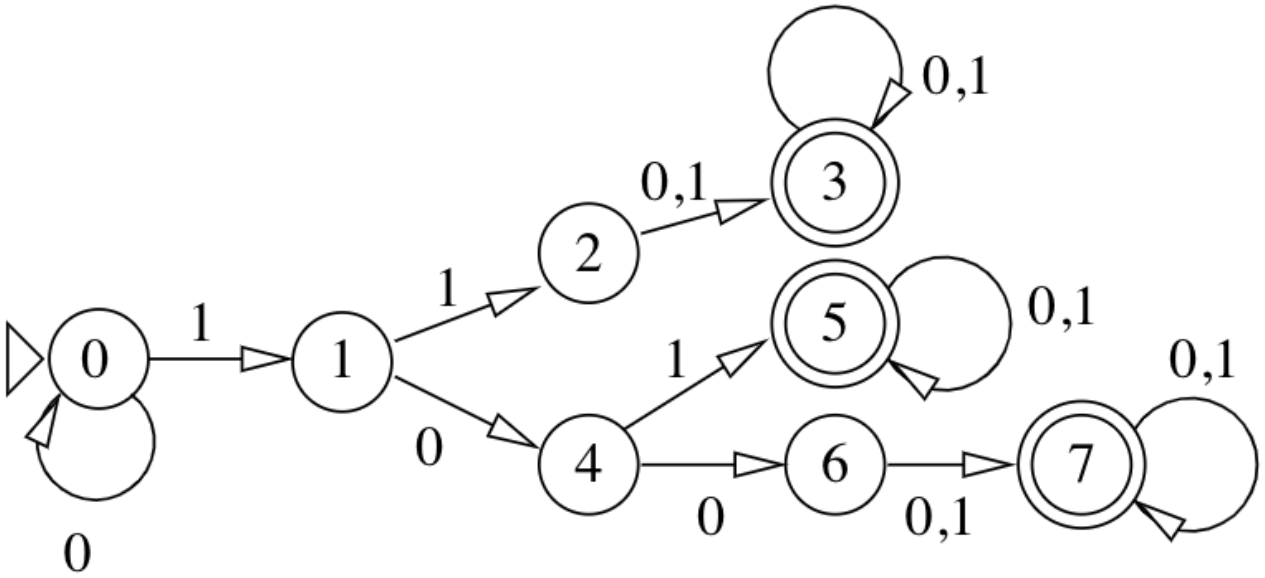
primo carattere a → stato 1; da qui: 2 per forza

se arriva a: stato finale ma: a in 5 o 8 → rifiuto

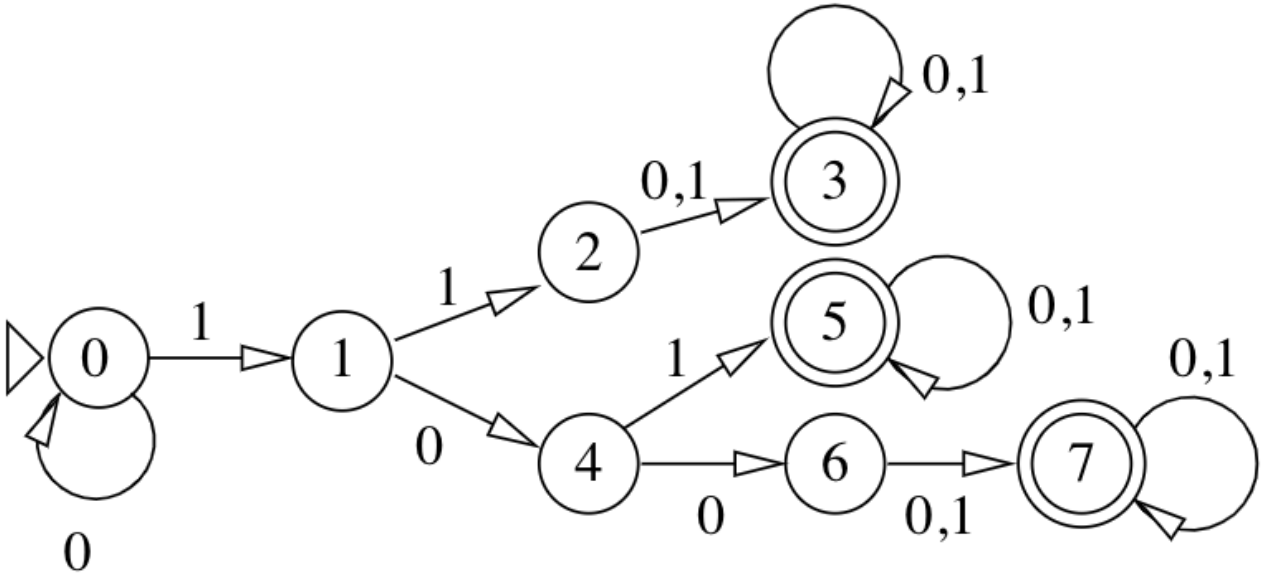
cicli su stati finali: va bene qualsiasi cosa arrivi dopo

Esercizio

- Caratteri `0` e `1`
- Accetta sequenza se il valore in decimale del numero binario rappresentato dalla sequenza è > 4
  Ad esempio, accetta `101` e `1000`, rifiuta `100` e `11` (si noti che la stringa in ingresso all'automa va letta da sinistra a destra)



Spiegazione



Zeri iniziali non cambiano il valore -> cappio

Diramazione ad albero

Stringa che rappresenta valore >4 -> accetta

Caratteri successivi: il valore aumenta (sempre stato finale)

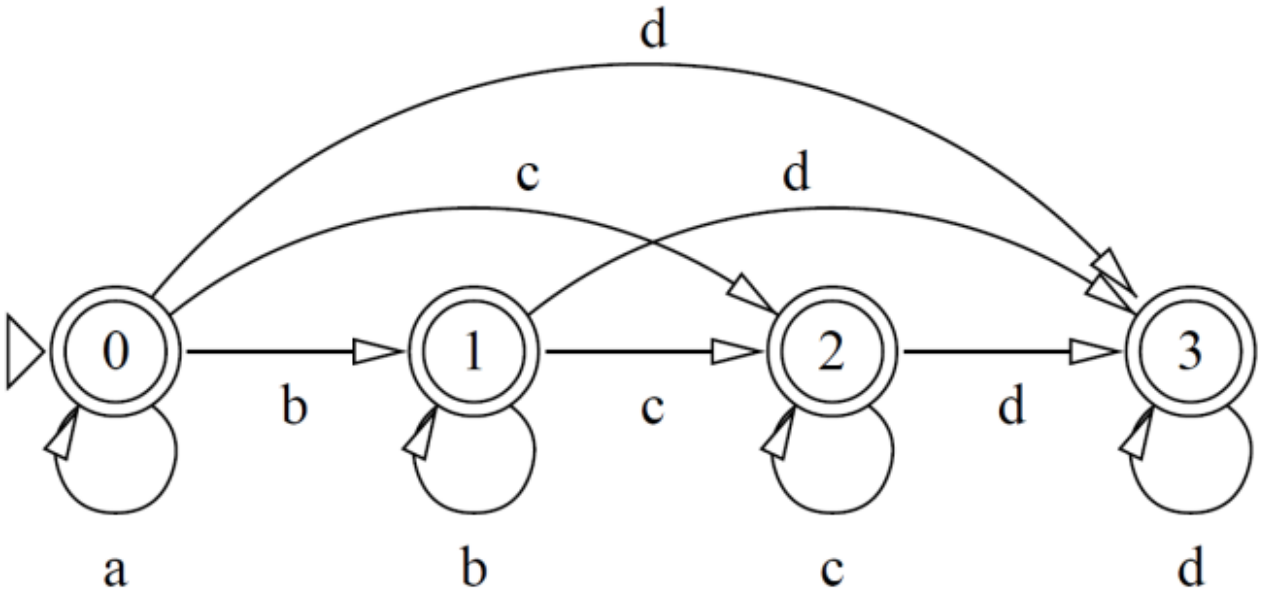
sequenze ordinate

stringhe di lettere {a,b,c,d}

accettare solo stringhe con lettere in ordine

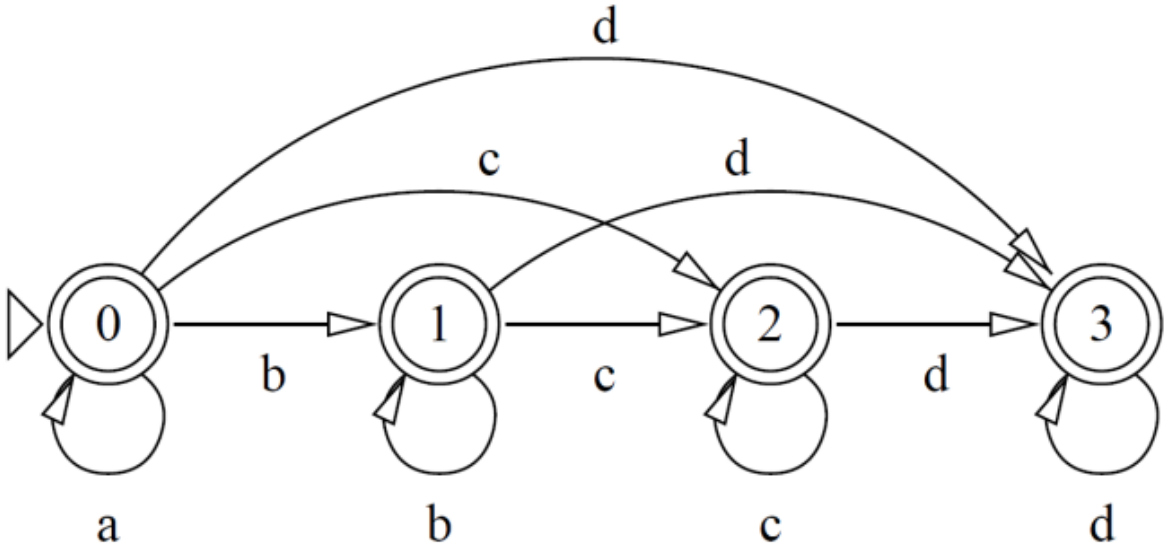
accetta per esempio aaaaa, accetta bbbcccc e abcd e aaaddd

non accetta aaaba e nemmeno abcda



progettazione automi: metodo

stato = condizione sulla sequenza arrivata finora



- in questo caso:

  stato 0 = lettere arrivate finora in ordine, ultima a

  stato 1 = lettere arrivate finora in ordine, ultima b

  stato 2 = lettere arrivate finora in ordine, ultima c

  stato 3 = lettere arrivate finora in ordine, ultima d

progettazione: stato successivo

stato 2 = lettere arrivate finora in ordine, ultima c

prossimo stato:

- nessuno (stringa rifiutata) se arriva a oppure b;
- stesso stato se arriva c
- stato 3 se arriva d

deriva dalla definizione di stato…

stato attuale e successivo

es: stato 2, arriva d:

- stato 2 = "sequenza precedente ok, ultimo carattere c"
- arriva d
- condizione attuale = "sequenza ok, ultimo d"

quindi: prossimo stato 3

progettazione degli automi: principio

assumendo una certa condizione vera (es. lo stato indica l'ultimo carattere di una sequenza in ordine), si fa in modo tale che all'arrivo del successivo carattere la condizione sia ancora vera

per *induzione*, la condizione è sempre vera

in generale: condizione = definizione dello stato

progettazione degli automi: in pratica

- si stabiliscono quali condizioni vanno distinte
- per ognuna si realizza uno stato
- sulla base delle condizioni s e q si stabilisce se

$$s \xrightarrow{c} q$$

- si modifica l'automa
- si rappresentano ingressi, uscite e stati in binario
- si scrive la funzione di stato successivo
- si scrive la funzione che distingue gli stati accettanti
- si sintetizzano queste funzioni con porte logiche

sintesi: modifica automa

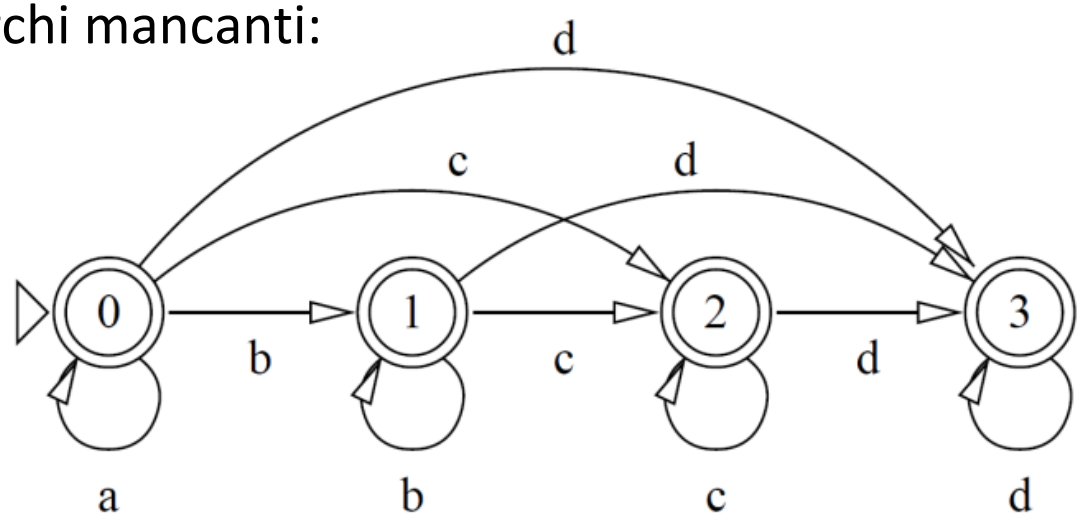
Obiettivo: modificare l'automa in modo che non manchi alcun arco uscente, cioè in ogni stato ci sia un arco uscente per ogni possibile carattere

- si introduce un singolo stato non finale p
- se da s manca l'arco con c, se ne mette uno che va in p
- p ha un cappio per tutti i caratteri

Esempio

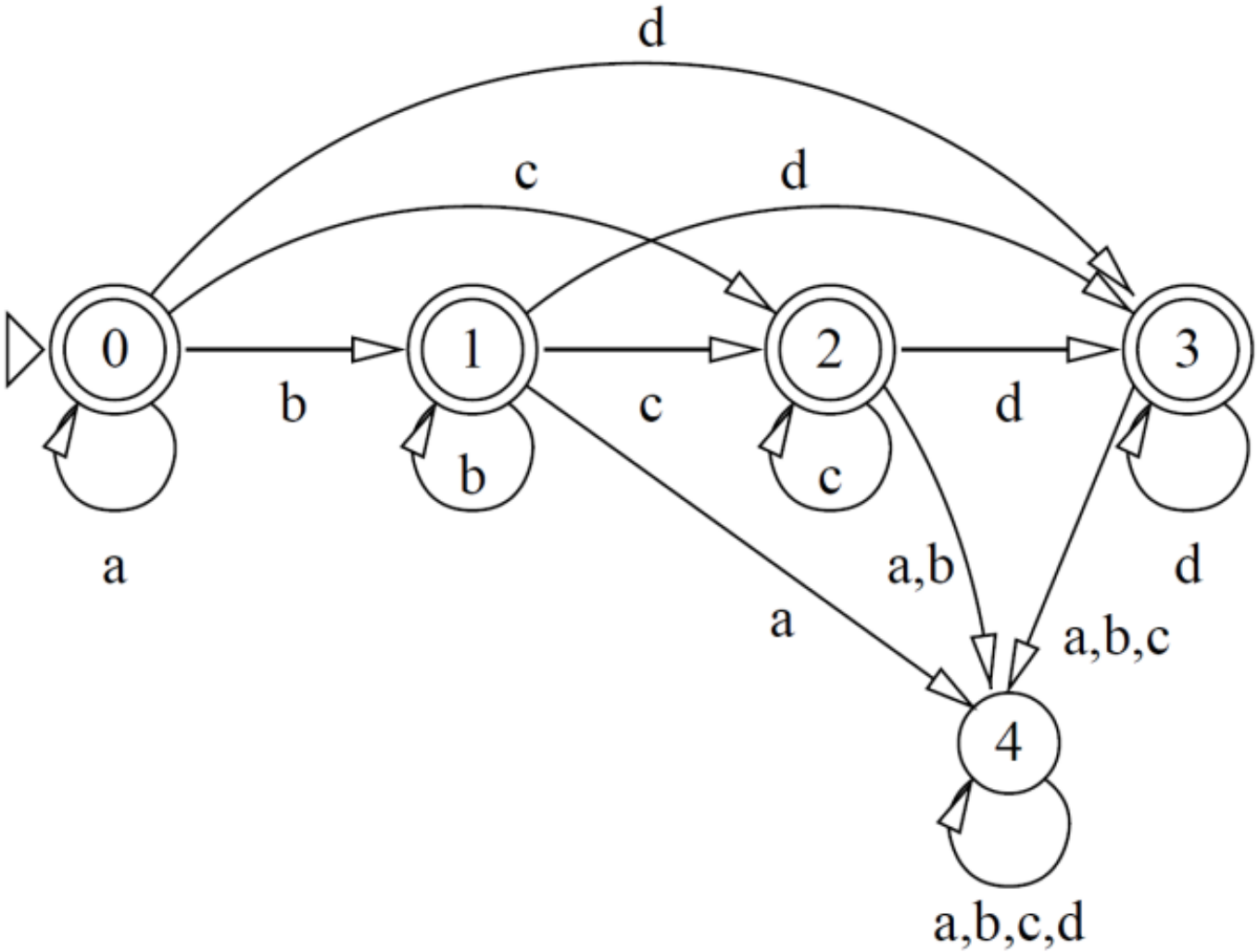
nuovo stato 4, qui vanno gli archi mancanti:

- da 1 con ingresso a
- da 2 con ingressi a e b
- da 3 con ingressi a, b e c



Con questa modifica la funzione di stato successivo sarà sempre definita

automa modificato



funzione di stato successivo

dati: stato e carattere

calcola: stato successivo

sempre definita (automa modificato)

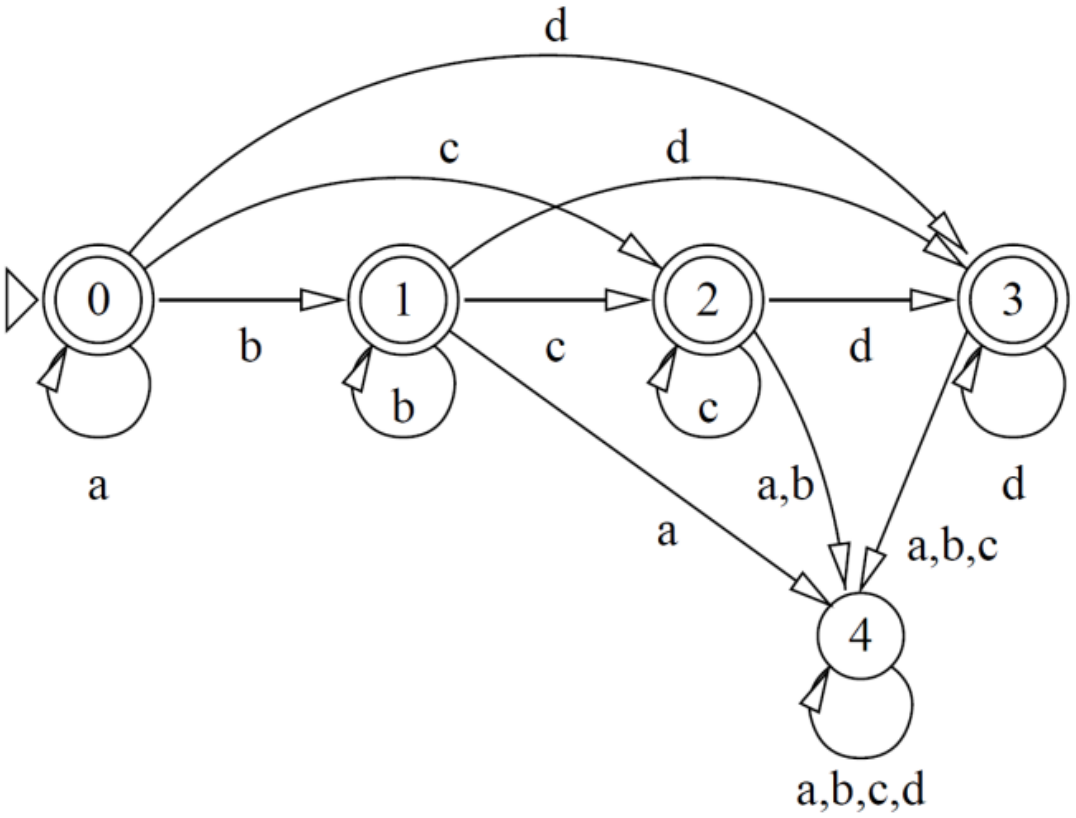
tabella: una riga per ogni stato e carattere

Stato	Carattere	Successivo

tabella stato successivo

una riga per ogni stato e
ogni carattere

Stato	Carattere	Successivo
0	a	0
0	b	1
0	c	2
0	d	3
1	a	4
1	b	1
1	c	2
1	d	3
…	…	…



codifica in binario

stati 0,1,2,3,4

→ servono tre bit

caratteri a,b,c,d

→ bastano due bit:

- a=00
- b=01
- c=10
- d=11

tabella, in binario

0,1,2,3,4 e a , b , c , d si mettono in binario

st	car	succ
0	a	0
0	b	1
0	c	2
0	d	3
1	a	4
1	b	1
1	c	2
…	…	…

→

st	car	succ
000	00	000
000	01	001
000	10	010
000	11	011
001	00	100
001	01	001
001	10	010
…	…	…

ogni bit di **succ** si sintetizza con un circuito

sintesi

Consideriamo solo il primo bit

si tolgono il secondo e il terzo bit da succ

resta una tabella con una uscita

solito sistema:

- righe con uscita 1 → and degli ingressi
- or di tutti questi

st	car		succ
000	00	→	000
000	01	→	001
000	10	→	010
000	11	→	011
001	00	→	100
001	01	→	001
001	10	→	010
…	…		…

stati finali

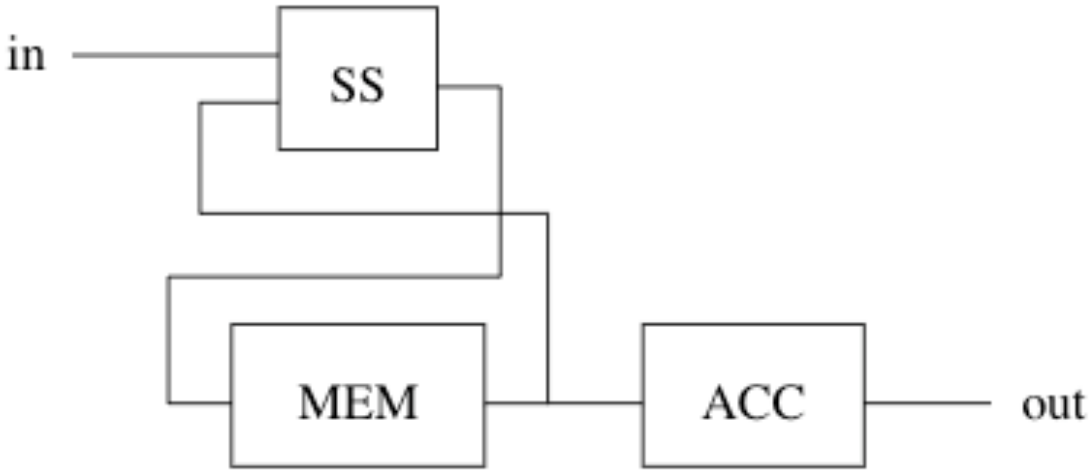
accetta=1, rifiuta=0

st		finale
000	→	1
001	→	1
010	→	1
011	→	1
100	→	0

Per ogni stato dice se è finale

La sintesi avviene nel solito modo (si prendono le righe in cui c'è 1, ecc.)

Circuito complessivo



SS

i tre circuiti dello stato successivo (visto)

ACC

il circuito che dice se uno stato è finale (visto)

MEM

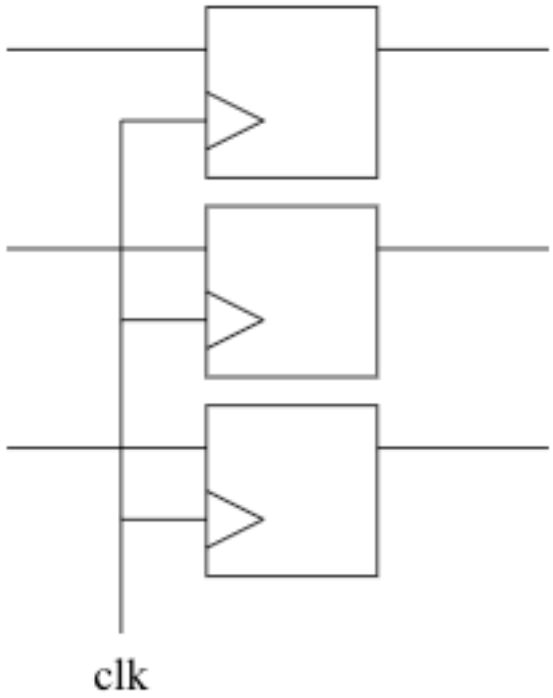
la memoria di stato

memoria=?

memoria

devo memorizzare tre bit (lo stato)

uso tre flip-flop



`clk` = clock

dice quando memorizzare lo stato successivo

definizione formale di automa

Ora diamo la definizione in termini matematici (serve ad esempio per dimostrare proprietà degli automi)
Gli ingredienti fondamentali sono:

- ingressi
- stati

Poi si deve definire lo stato iniziale, gli stati successivi, e gli stati finali.

Indichiamo con

I insieme degli ingressi
S insieme degli stati

Negli esempi precedenti:

- Sequenze ordinate: **I**={a , b , c , d} e **S**={0,1,2,3}
- Valore sequenza binaria > quattro: **I**={0 , 1} e **S**={0,1,2,3,4,5,6,7}

stati iniziale e finali

insieme di tutti gli stati: **S**

stato iniziale:

s_0 è un elemento di **S**

insieme degli stati finali (doppio cerchio): **A**

A è un sottoinsieme di **S**

dal punto di vista matematico:

$$s_0 \in \mathbf{S} \text{ e } \mathbf{A} \subseteq \mathbf{S}$$

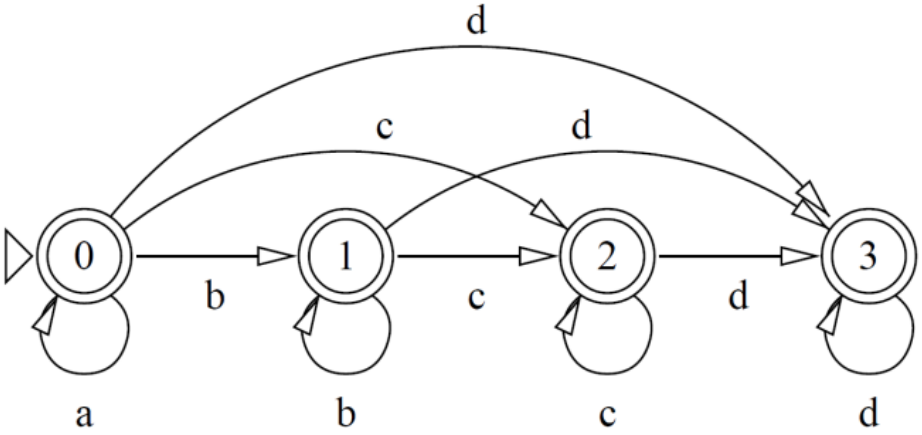
stati: esempi

Sequenze ordinate:

S={0,1,2,3}

s_0=0

tutti stati finali: **A**={0,1,2,3}

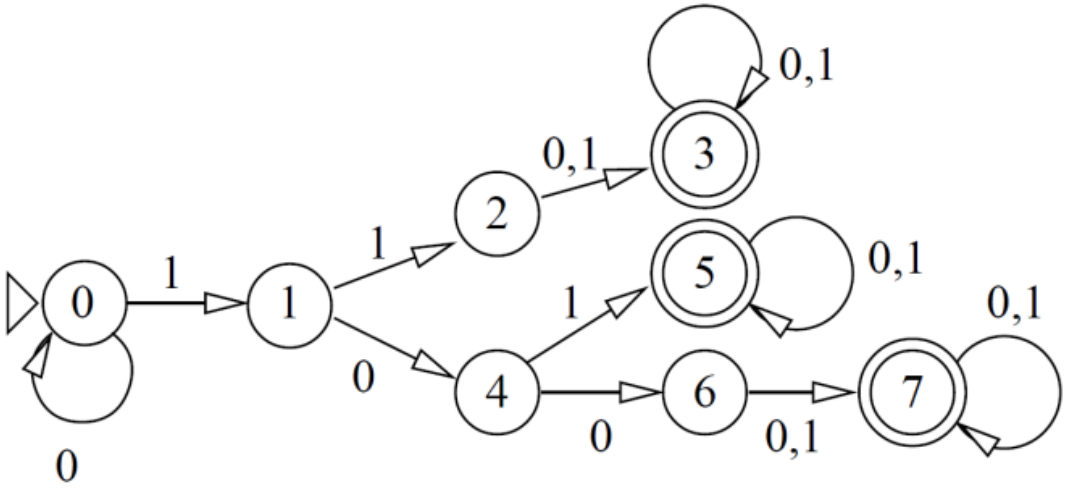


valore > quattro:

S={0,1,2,3,4,5,6,7}

s_0=0

stati finali: **A**={3,5,7}



stato successivo

da stato *s* e ingresso `i` si passa a un nuovo stato

è una funzione matematica:

$$N : \mathbf{S} \times \mathbf{I} \rightarrow \mathbf{S}$$

permette di calcolare lo stato successivo

N(*s*,`i`)=*s'* vuol dire:

se in *s* arriva `i`, si va in *s'*

stato successivo: esempio

da 0 con c si va in 2

quindi: N(0,c)=2

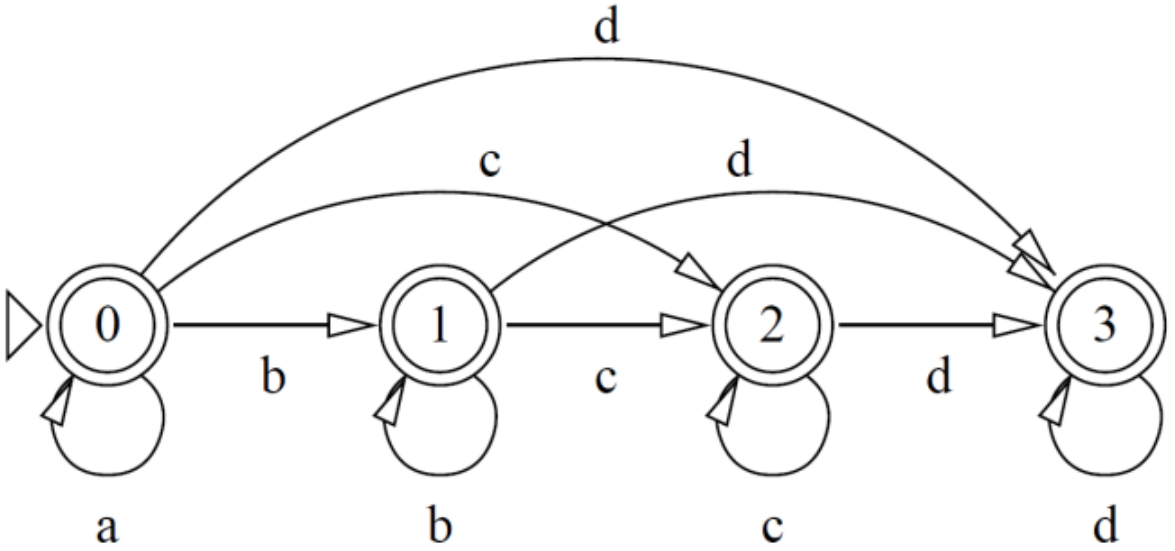
e poi:

N(2,d)=3

N(0,a)=0

N(3,a) indefinito

ecc.



funzione e tabella

N : **S** × **I** → **S**

dati *s* e `i` fornisce lo stato successivo N(*s*,`i`)

Si noti che nella tabella vista in precedenza e riportata al lato

si guarda la riga con *s* e i per ottenere

lo stato successivo riportato nella

terza casella della riga.

sono di fatto la stessa cosa

Stato	Carattere	Successivo
0	a	0
0	b	1
0	c	2
0	d	3
1	a	4
1	b	1
1	c	2
…	…	…

automa: definizione completa

un automa è una quintupla $\langle \mathbf{I}, \mathbf{S}, s_0, \mathrm{N}, \mathbf{A} \rangle$

I un insieme, detto insieme degli ingressi

S un insieme, detto insieme degli stati

s_0 un elemento di S, detto stato iniziale

A un sottoinsieme di S, detto insieme degli stati finali

N una funzione $\mathbf{S}\times\mathbf{I}\rightarrow\mathbf{S}$, detta funzione di stato successivo

definizioni in termini di insiemi

già quasi tutto definito in termini di:

- insieme
- elemento di un insieme
- sottoinsieme

tranne: funzione N

prodotto cartesiano:

S×**I**×**S** = insieme di triple $\langle s,\mathtt{i},s'\rangle$ con $s\in$**S**, $\mathtt{i}\in$**I** e $s'\in$S

funzione (parzialmente definita):

N:**S**×**I**→**S** = sottoinsieme di **S**×**I**×**S** che non contiene due o più terne che hanno gli stessi primi due elementi

intuitivamente: $\langle s,\mathtt{i},s'\rangle$ se $N(s,\mathtt{i})=s'$

Queste definizione di N in termini di insiemi servirà in seguito per definire gli automi non deterministici.

stato dopo una stringa

stringa $c_1 c_2 \ \ldots \ c_{n-1} c_n$

porta allo stato:

$N(N(N(\ \ldots N(N(s_0,c_1),c_2) \ \ldots\ ),c_{n-1}),c_n)$

- stato iniziale s_0
- primo carattere c_1, porta a $N(s_0,c_1)$
- secondo carattere c_2, porta a $N(N(s_0,c_1),c_2)$
- terzo carattere, ecc.

carattere $c_i \rightarrow$ applicare $N(\ldots,c_i)$

carattere $c_n \rightarrow N(N(N(\ \ldots N(N(s_0,c_1),c_2) \ \ldots\ ),c_{n-1}),c_n)$

accettazione

$$N(N(N(\ \ldots\ N(N(s_0,c_1),c_2)\ \ldots\ ),c_{n-1}),c_n)$$

- se questo è uno stato finale, la stringa $c_1c_2\ \ \ldots\ \ c_{n-1}c_n$ è accettata
- altrimenti è rifiutata

$$c_1c_2\ \ \ldots\ \ c_{n-1}c_n \text{ accettata}$$

se e solo se

$$N(N(N(\ \ldots\ N(N(s_0,c_1),c_2)\ \ldots\ ),c_{n-1}),c_n) \in \mathbf{A}$$

vincolo sulla funzione

N non contiene due triple $\langle s,\mathtt{i},s'\rangle$ con gli stessi s e $\mathtt{i}$

$\langle s,\mathtt{i},s'\rangle \in$ N vuol dire:
da stato s con ingresso $\mathtt{i}$ si va a stato s'

no due triple con stessi s e $\mathtt{i}$ → stato successivo unico

più triple con stessi s e $\mathtt{i}$ → più stati successivi

automi non deterministici

deterministici

$s, \mathtt{i} \rightarrow s'$

non deterministici

$s, \mathtt{i} \rightarrow s', s'', \ldots$

stando in uno stato, arriva un carattere: si può andare in più stati successivi

cioè, in che senso?

sistemi teorici e reali

reali

dato uno stato, arriva un input: si va in un altro stato

teorici

volendo, si può immaginare di andare in più stati diversi

sempre con un singolo carattere di input

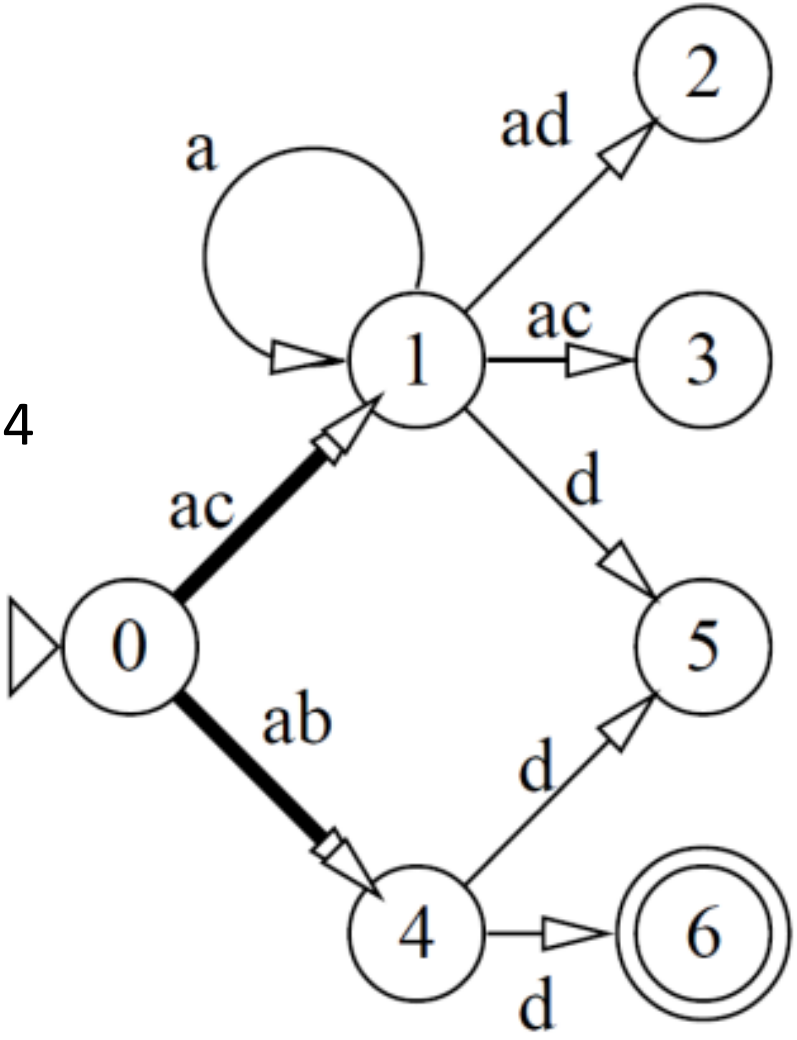
automa non deterministico: esempio

si inizia in 0

arriva a

si può andare in 1 oppure in 4

dove vado?

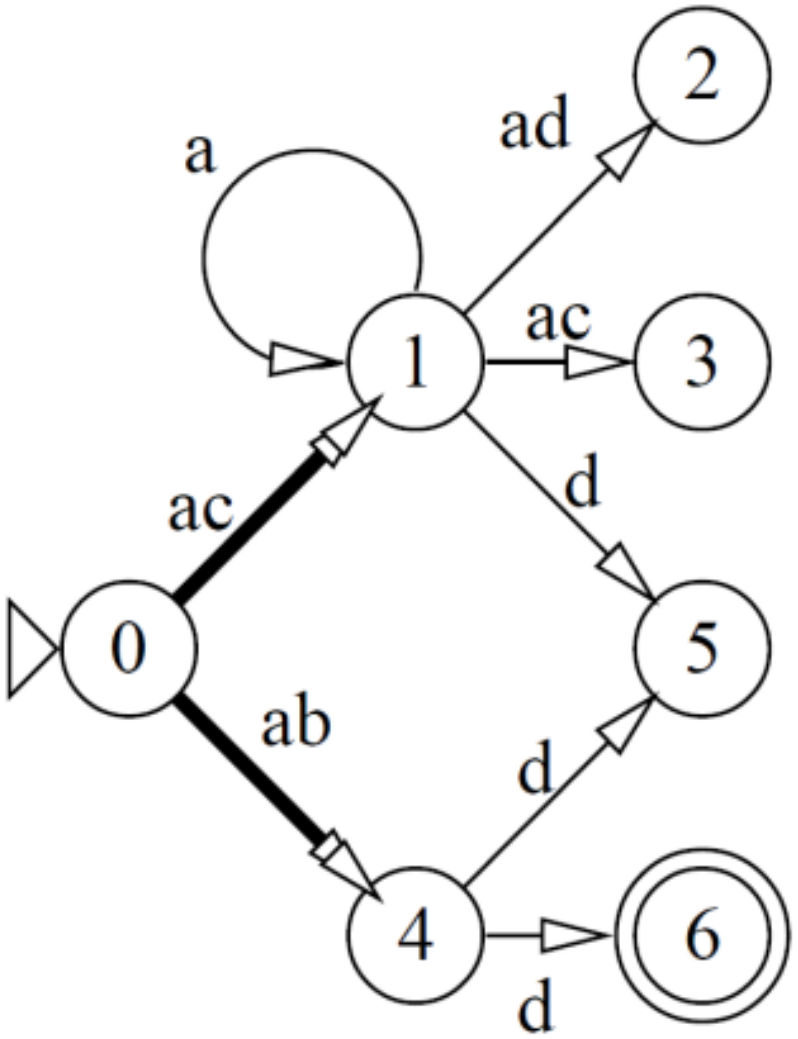


non determinismo: dove vado?

da 0 con a: vado in 1 o in 4?

si fanno entrambe le scelte

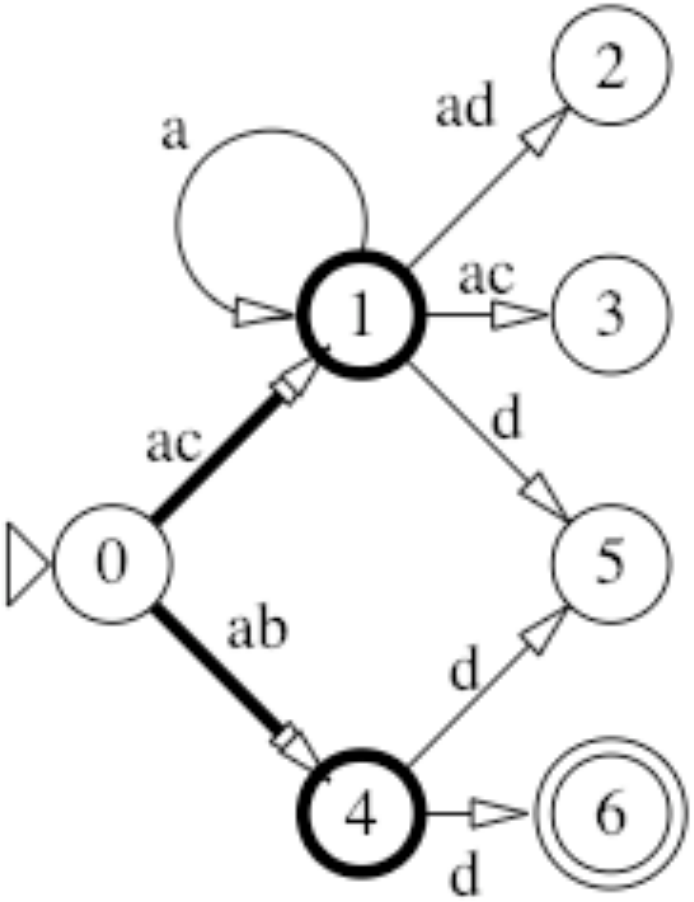
si finisce in 1 e in 4 insieme



due stati insieme

vado in 1 e anche in 4

ora arriva d



da due stati, dove vado?

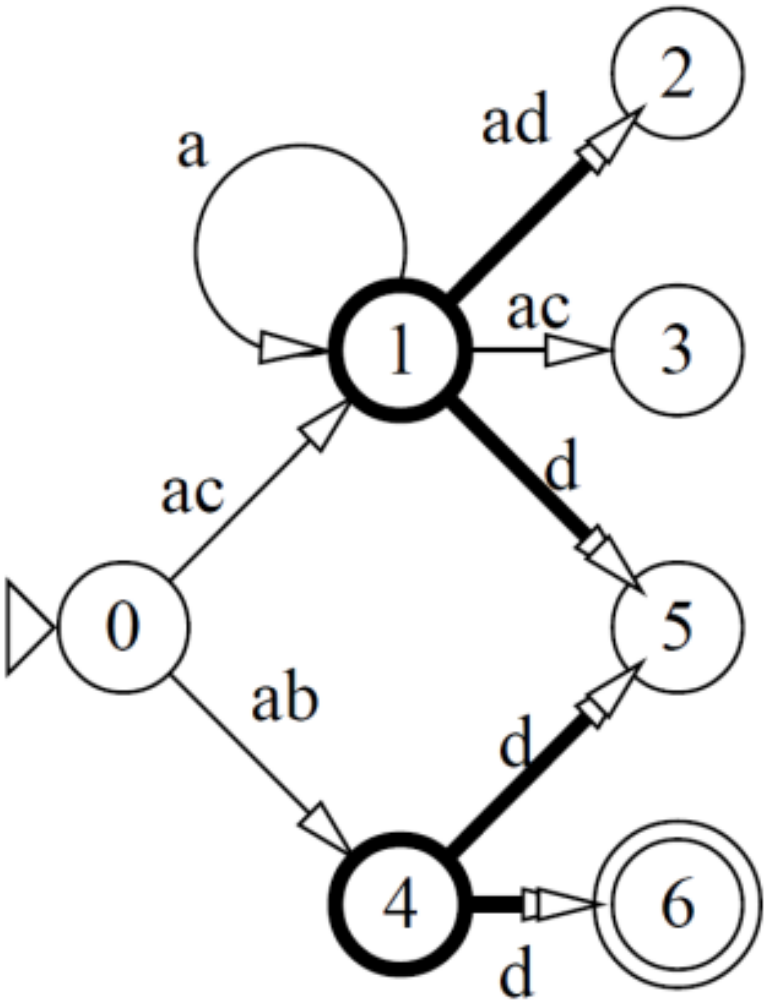
sono in 1 e 4

con d posso andare:

da 1 in 2 e 5

da 4 in 5 e 6

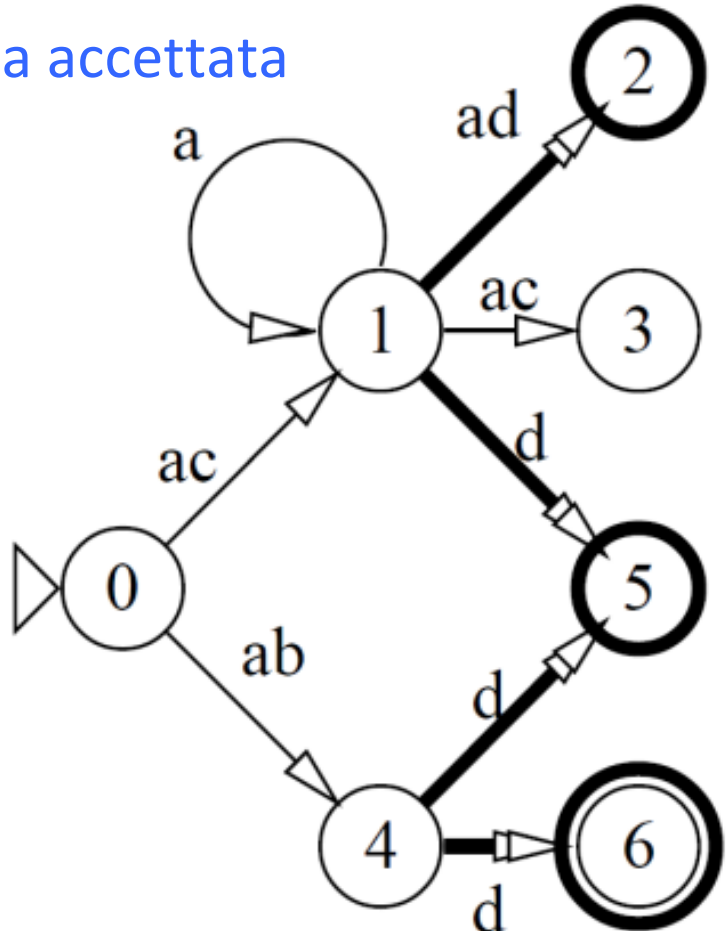
vado in tutti questi



stati dopo due caratteri

arrivo in 2, 5 e 6

uno di questi **è finale** → stringa accettata



ramificazioni, riassunto

esempio: stringa ad

- da 0, con a: si va in 1 o in 4
- arriva d:
  - da 1 si va in 2 o in 5
  - da 4 si va in 5 o in 6

  vanno considerate tutte le possibilità

si arriva in 2, 5 e 6

6 finale → stringa ad accettata

percorsi possibili

si parte dallo stato iniziale

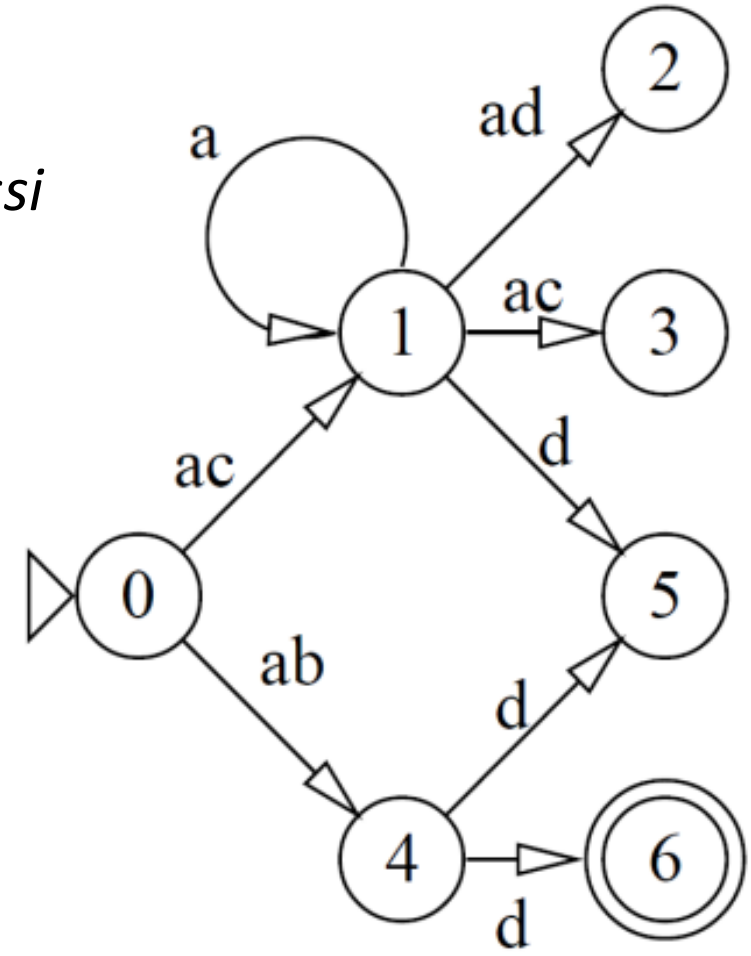
le frecce indicano *gli spostamenti ammessi*

la prima a permette di raggiungere 1 o 4

(nota: non permette di restare in 0)

c'è un percorso per uno stato finale

→ stringa accettata



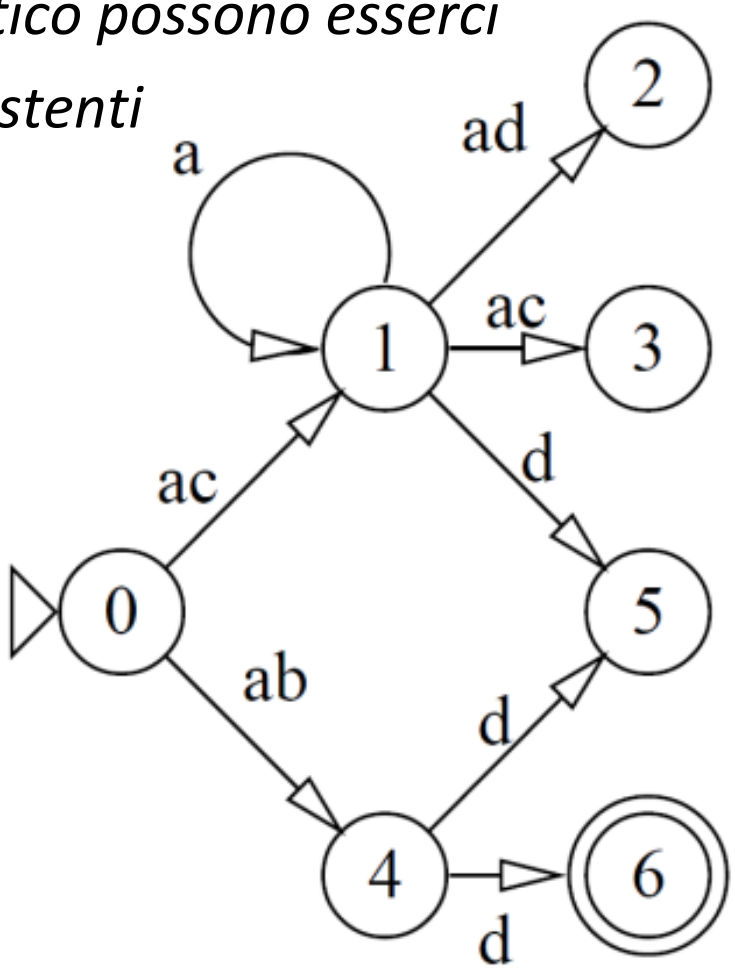
scelte obbligate o inesistenti

cc porta solo in 3, bc non porta da nessuna parte

in un automa non deterministico possono esserci

anche scelte obbligate o inesistenti

non solo scelte multiple



definizione formale

Automa non deterministico:

- uguale a quello deterministico , togliendo il vincolo su N
- È quindi una quintupla $\langle$ **I**, **S**, s_0, N, **A** $\rangle$
  con **I**, **S**, s_0, **A** uguali a quanto visto in precedenza, ma con N senza il vincolo sulle terne.
  Il vincolo è solo: $N \subseteq \mathbf{S}\times\mathbf{I}\times\mathbf{S}$

Approfondimento: tecnicamente, N è in questo caso una funzione che va dall'insieme **S**×**I** all'insieme delle parti di **S** (che è l'insieme di tutti i sottoinsiemi di **S**). Cioè $N:\mathbf{S}\times\mathbf{I}\rightarrow P(\mathbf{S})$

La "scelta inesistente" menzionata alla slide precedente corrisponde formalmente ad associare ad una coppia (s,`i`) appartenente a **S**×**I** l'insieme vuoto (che è contenuto nell'insieme delle parti di **S**). In questo caso cioè $N(s,\texttt{i})=\varnothing$

stato di arrivo di una stringa

stato attuale s → insieme di stati attuali **SA**

$$\text{successivo}(\mathbf{SA}, \mathtt{i}) = \{s' \mid \langle s,\mathtt{i},s'\rangle \in \mathrm{N} \text{ e } s \in \mathbf{SA}\}$$

stato di una stringa:

si applica *successivo* una volta per ogni carattere della stringa $c_1c_2\ldots c_{n-1}c_n$:

successivo(successivo(… successivo(successivo({s_0},c_1),c_2),…,c_{n-1}), c_n)

è un insieme di stati, se uno di questi stati in **A** → stringa accettata

vantaggi del non determinismo

si applica a varie cose, oltre agli automi

in generale:

non determinismo = varie scelte possibili a ogni passo

esempio: il problema dei 150 cavalieri

forma generale di molti problemi:

- scegliere dei valori
- verificare se soddisfano una condizione

non determinismo: formalizza la scelta

il problema dei 150 cavalieri

in questo caso:

- scegliere le posizione per ogni cavaliere
- verificare che non ci siano rivali vicini che non ci siano due cavalieri allo stesso posto che ogni cavaliere abbia un posto

soluzione non deterministica

- scegli posizione cavaliere 0
- scegli posizione cavaliere 1
- …
- scegli posizione cavaliere 149
- verifica posizioni e rivalità

se esiste una sequenza di scelte che porta a una soluzione, allora esiste una disposizione accettabile

programma

su un ipotetico «computer non deterministico» (che non esiste)

```
for cavaliere in range(0, 150):
   posizione[cavaliere]=scelta(0, 1, ... , 149)
for c in range(0, 150):
   for d in range(0, 150):
      if rivali[c][d]:
         if posizione[c]==(posizione[d]+1)%150:
            return false;
return true;
```

Nota: Per semplicità assumiamo che la funzione `scelta(0,..,149)` assegni posizioni ai cavalieri senza mettere contemporaneamente due cavalieri nello stesso posto

risultato?

scelte arbitrarie delle posizioni

esiste una sequenza di scelte che porta a una soluzione

⇓

la soluzione esiste

come per gli automi:

esiste percorso per stato finale ⇒ stringa accettata

i sistemi non deterministici non esistono nella realtà, si simulano con sistemi deterministici. Vantaggio del non determinismo:

il non determinismo permette di codificare alcuni problemi complessi in modo semplice

li codifica, non li risolve

non-determinismo: riassunto

- il non determinismo è un meccanismo teorico che permette di esprimere diversi problemi in modo facile,

ma

- gli elaboratori reali sono deterministici, per cui i sistemi non deterministici vanno convertiti in deterministici in qualche modo.

espressioni regolari e automi

applicazione degli automi non deterministici

nel verificare una stringa, si possono presentare scelte

Nota: si sta parlando del modo che usano i programmi per vedere se una stringa collima con un'espressione regolare

es: come l'interprete Python esegue `re.match`

espressioni regolari con scelte

- espressione regolare `r.*doc`
  = `r` seguita da caratteri qualsiasi e poi `doc`
- stringa `ridotto.doc`

la prima `d` è parte di `.*`

la seconda è l'inizio di `doc`

```
ridotto.doc
↑-------↑↑↑
r   .*  doc
```

esecuzione

leggo i caratteri uno per volta:

- primo carattere `r` ⇒ `r.*doc`
- carattere `i` ⇒ `r.*doc`
- carattere `d` ⇒ `r.*doc` ma anche `r.*doc`
- carattere `o`: stessa cosa
- carattere `t`: solo ora so che siamo ancora in `.*`
- …

riconoscimento in Python

Di fronte a due possibilità se ne sceglie una

ma ci si ricorda dell'altra

se va male, si prova l'altra

questo algoritmo è però semplicistico

algoritmo "astuto"

si pone una scelta fra:

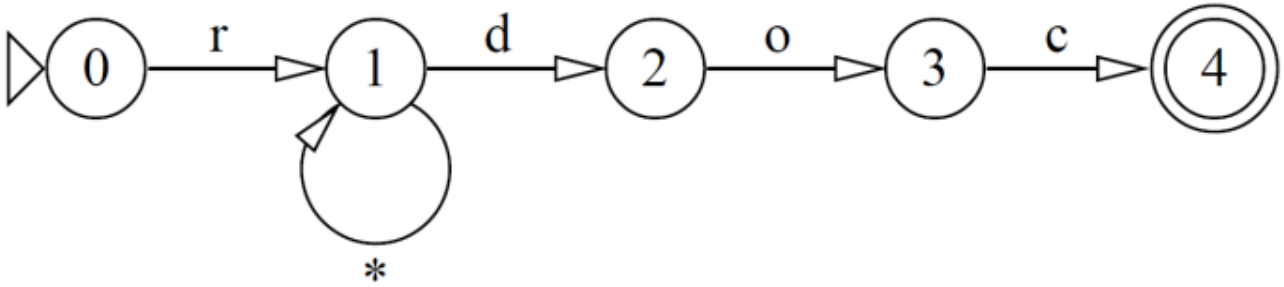
- `d` è parte di `.*`
- `i=.*` e `d` è la prima lettera di doc

si verifica se almeno una scelta porta al successo

in questo caso: la prima scelta

espressione → automa

esistono scelte: automa non deterministico



in 1 con carattere d:

- si resta in 1
- si va in 2

stesso stato e carattere, ma due successori: automa non deterministico

espressione → automa

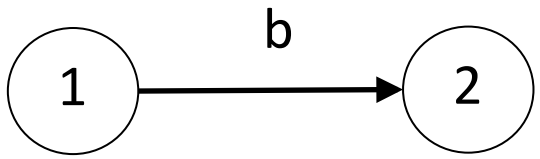
modo automatico:

si parte dalle sottoespressioni più semplici e le si compone via via, fino a ottenere l'automa della formula intera

esempio

`((bd|c)a?)+`

vogliamo l'automa. Partiamo da quello che riconosce b da solo:



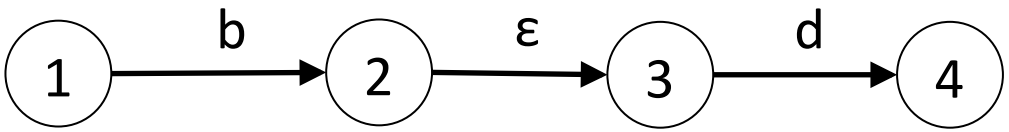
Uno simile per a, per c e per d

poi: si compongono…

composizione: concatenazione

fatti: automi per b e per d

automa per bd:



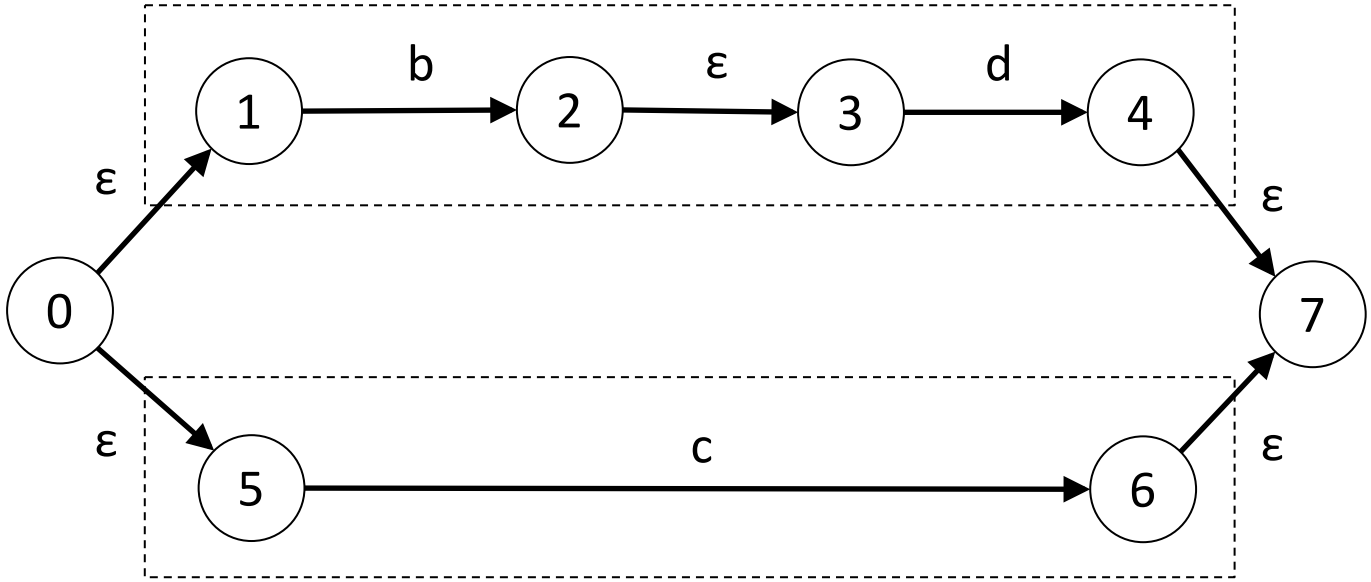
volendo: eliminare arco ε

fondere i due stati in mezzo

composizione: alternativa

abbiamo l'automa per bd e quello per c

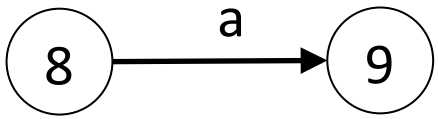
automa per bd|c:



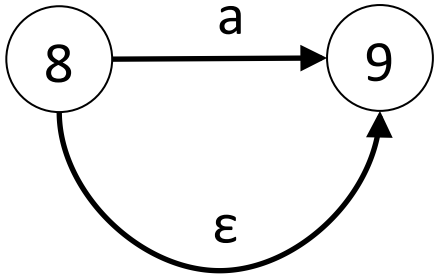
sono i due messi in parallelo

composizione: opzionalità

l'automa di a è come quello di b:



a? è a opzionale:



arco ε da iniziale a finale

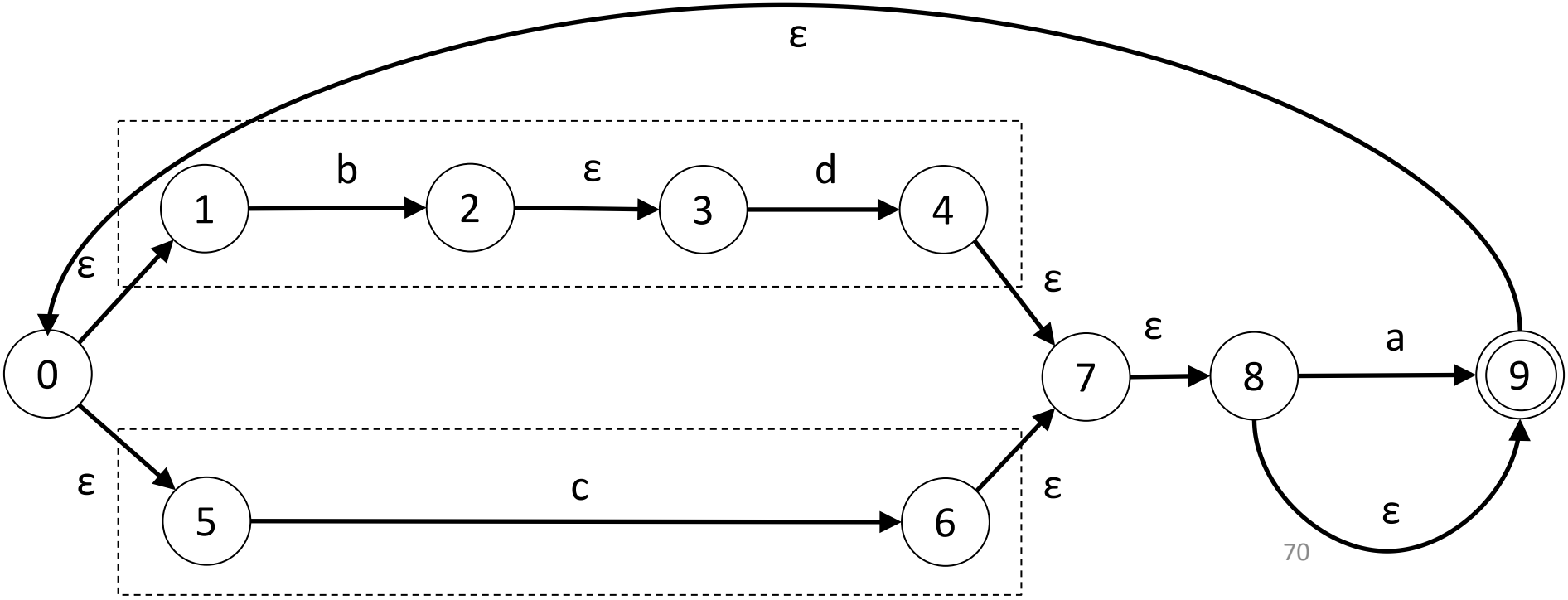
composizione

automi di bd|c e di a? noti

si concatenano e si ottiene quello di (bd|c)a?

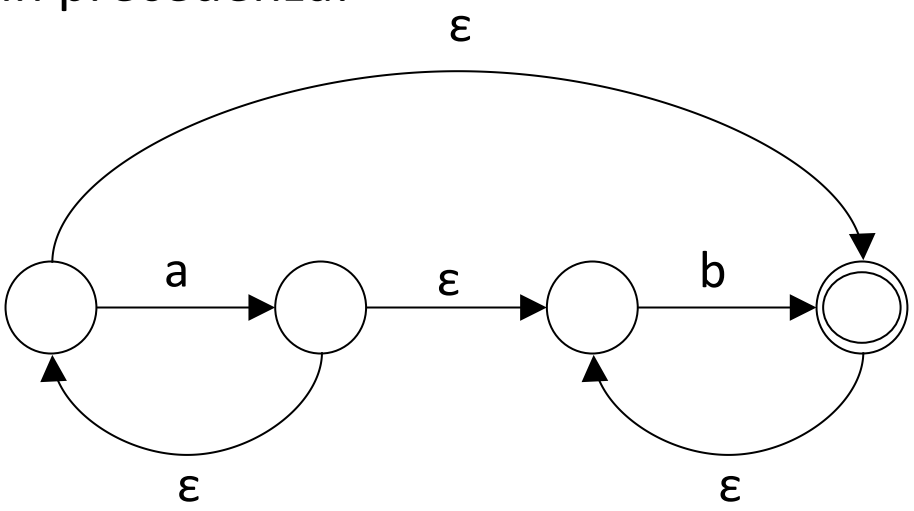
ora: ripetizione non nulla di tutto (sarebbe il +)

dato: automa di (bd|c)a? → automa di ((bd|c)a?)+:



Problema

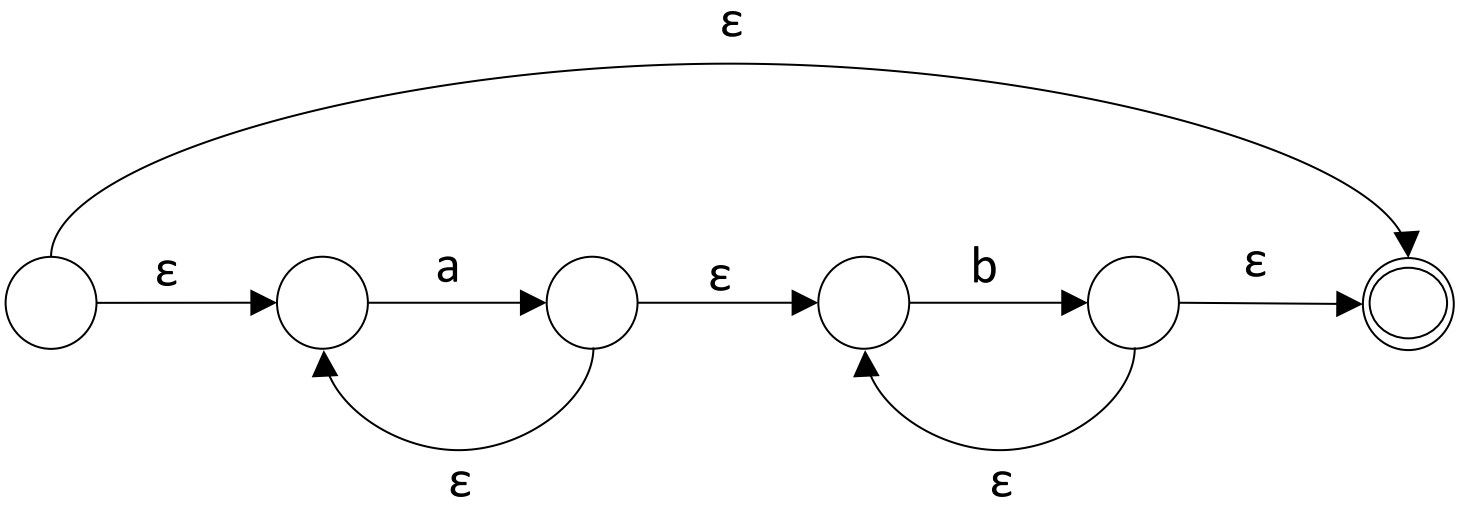
- In generale è però necessario aggiungere un ulteriore stato che diventa quello iniziale ed un ulteriore stato, che diventa quello finale.
- Si consideri ad esempio l'automa di (a+b+)? Realizzato con la stessa tecnica vista in precedenza:



l'automa di (a+b+)? Così realizzato accetterebbe anche aaa

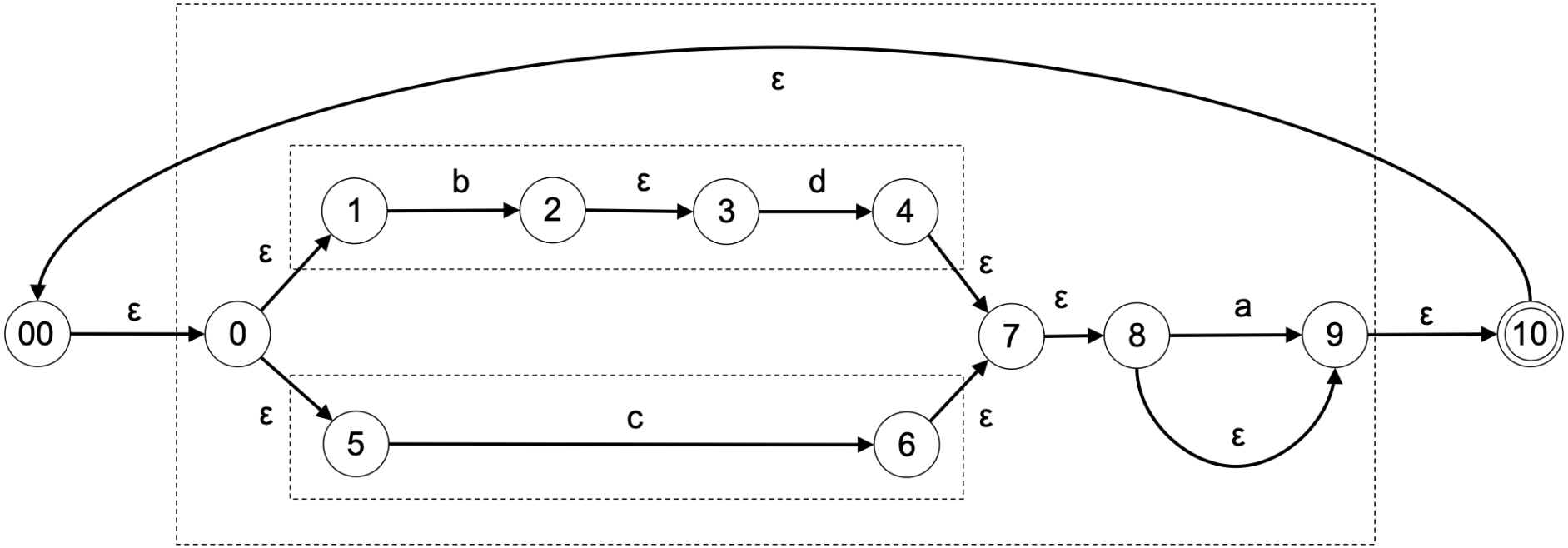
Aggiungere uno stato finale ed uno stato iniziale

Versione corretta dell'automa che riconosce le stringhe che collimano con l'espressione regolare (a+b+)?



Vale come regola generale (anche se potrebbe non servire)

Versione completa dell'automa per ((bd|c)a?)+



Ricapitolazione

A* è uguale a (A+)?

automi di + e ?: visti sopra

espressione → automa non deterministico

si procede sempre nello stesso modo:

- si parte dagli automi dei singoli caratteri
- si compongono nel modo visto

si ottiene un automa non deterministico

ora: convertirlo in deterministico

conversione da non deterministico a deterministico

espressione regolare → automa non deterministico → automa deterministico

deterministico

in ogni momento, un solo stato

non deterministico

in ogni momento, anche più stati possibili

principio della conversione

deterministico: in ogni momento un solo stato

non deterministico: più stati in ogni momento

(dato che ci sono le scelte)

idea: insieme_stati → stato

l'automa non deterministico si trova in ogni momento in un insieme di stati

per ogni insieme di stati, si crea uno stato nell'automa deterministico

Stati

più precisamente:

se successore può portare a un insieme di stati **S**, allora l'automa deterministico deve avere uno stato per **S**

esempio:

- se lo stato è {1, 2}
- se in 1 il carattere a porta a 4;
- se in 2 il carattere a porta a 5, 8, 9;
- allora a porta da {1,2} a {4, 5, 8, 9}.

ogni stato dell'automa deterministico simula un insieme di stati dell'automa non deterministico

le frecce sono di conseguenza

Stati finali

esiste un percorso verso uno stato finale

⇓

stringa accettata

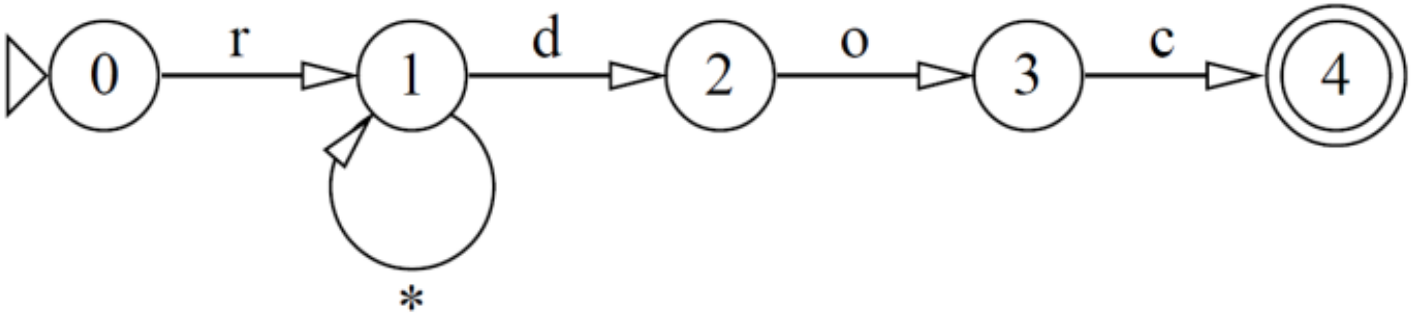
quindi

gli stati finali dell'automa deterministico sono quelli che corrispondono a insiemi che contengono almeno uno stato finale dell'automa non deterministico

basta un solo stato finale a rendere finale l'intero insieme

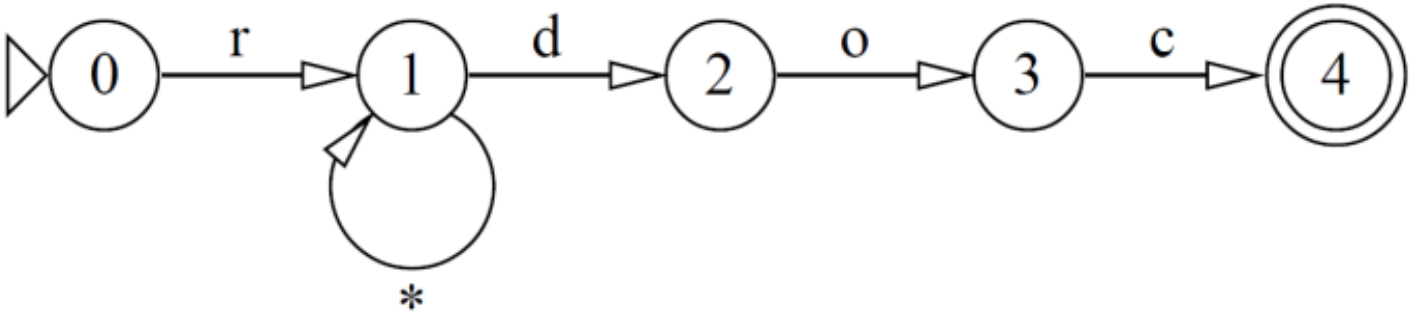
tornando all'esempio

- In quali insiemi di stati può trovarsi?



- da {0} solo in {1}
- ma da 1 in 1 o in 2
- quindi: stato {1,2}
- ancora…

Insieme di stati

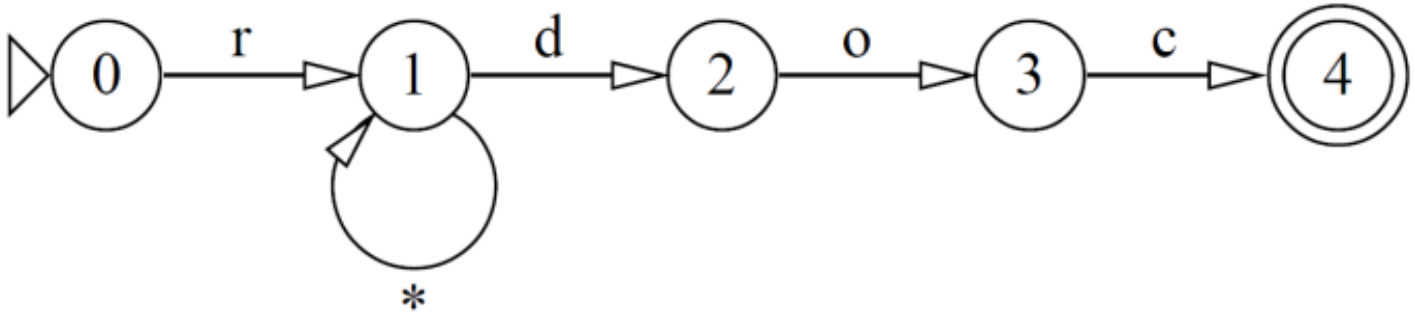


da {1,2} dove si va?

- con d: 1→1, 1→2, 2→niente insieme {1,2}
- con o: 1→1, 2→3 insieme {1,3}
- altro: 1→1, 2→niente insieme {1}

serve lo stato {1,3}

Da stato {1,3}

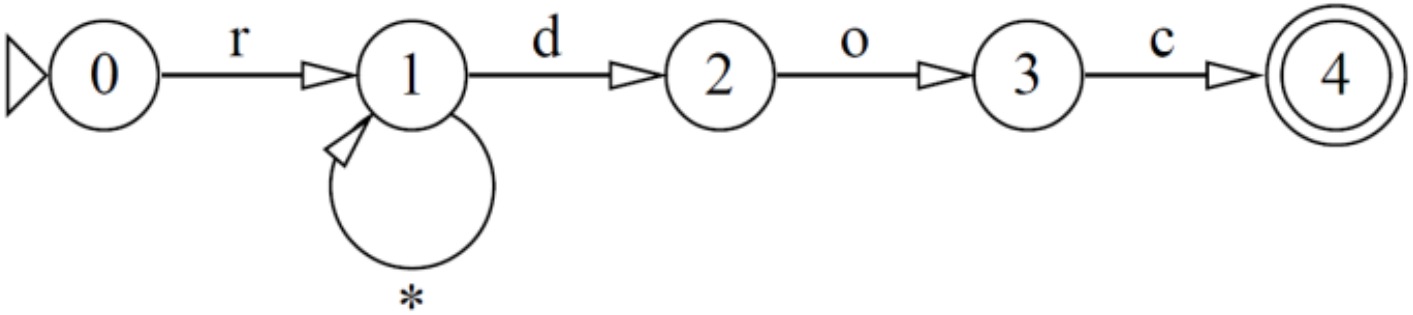


stessa cosa: c porta a {1,4}

d porta in {1,2}

tutti gli altri in 1

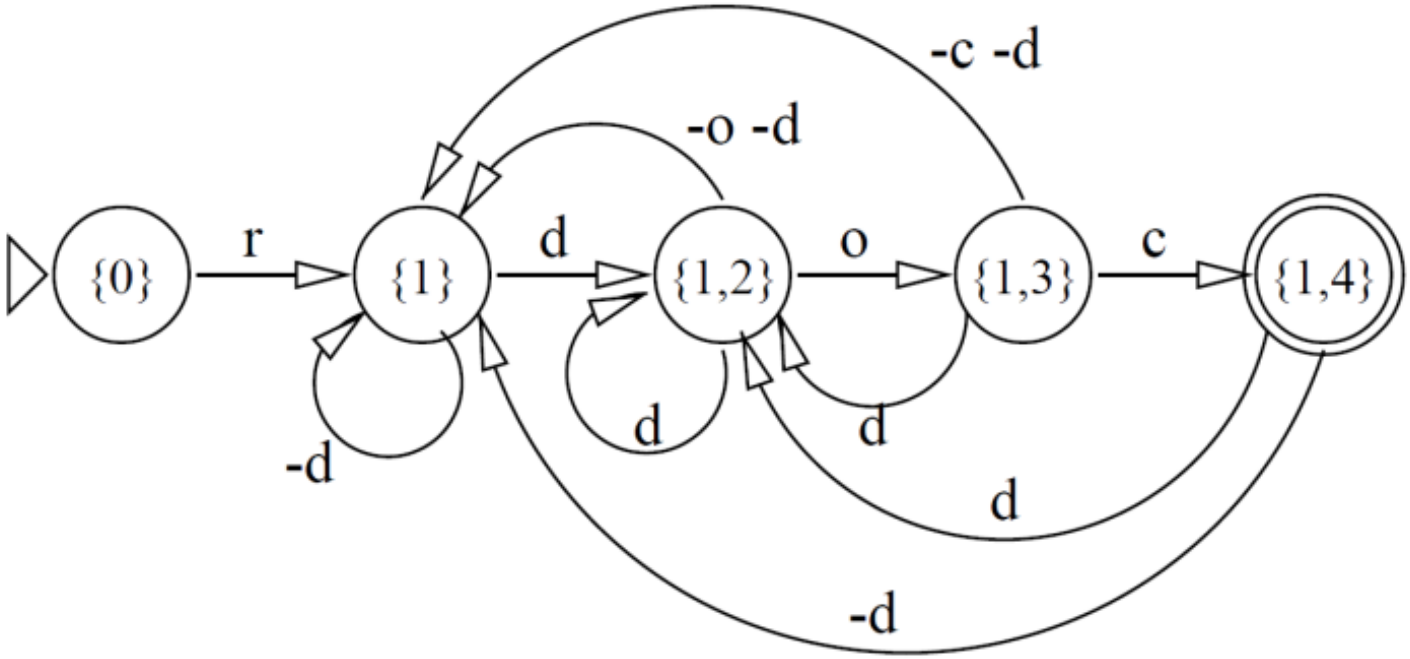
Stato {1,4}



d porta in 1 e in 2

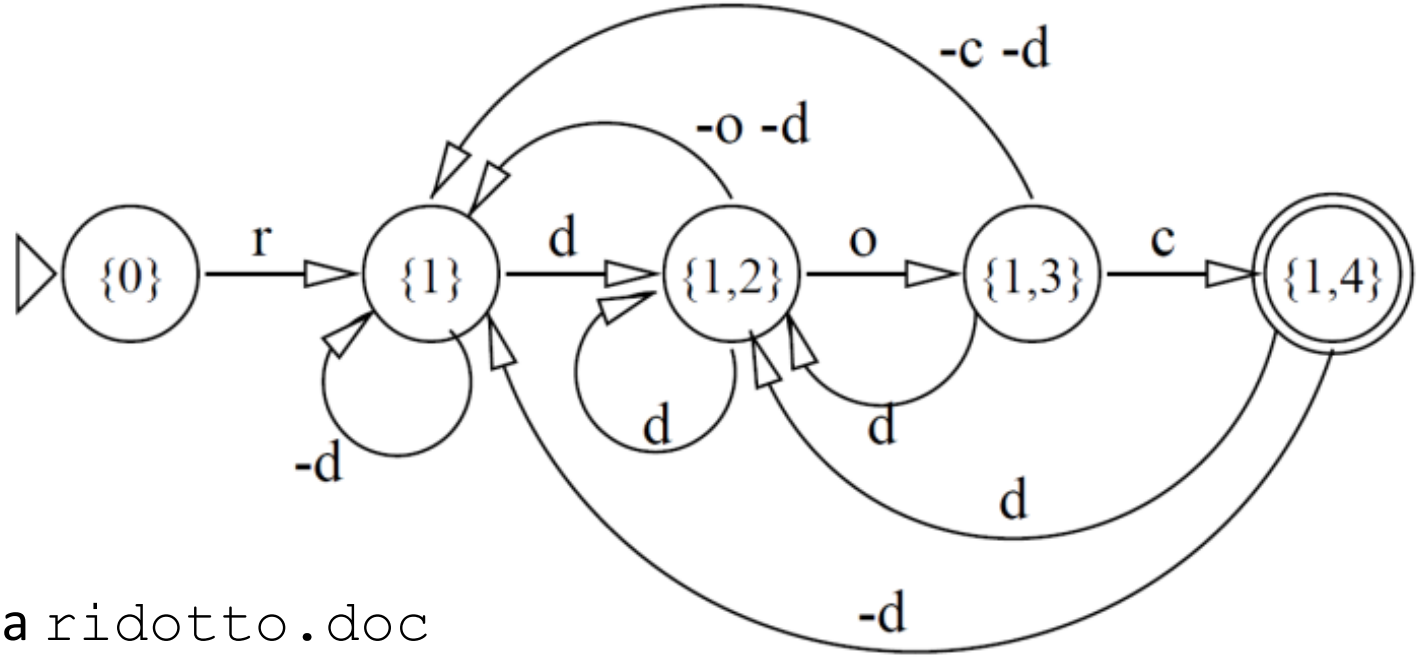
tutti gli altri in 1

Automa complessivo



-d = qualsiasi carattere tranne d

Prova esecuzione



stringa `ridotto.doc`

- `rido` → {1,3}
- `t` → {1}
- `to.` → {1}
- `doc` → {1,4}

stringa accettata

automi con output

quelli visti finora finivano con:

- stringa accettata
- stringa rifiutata

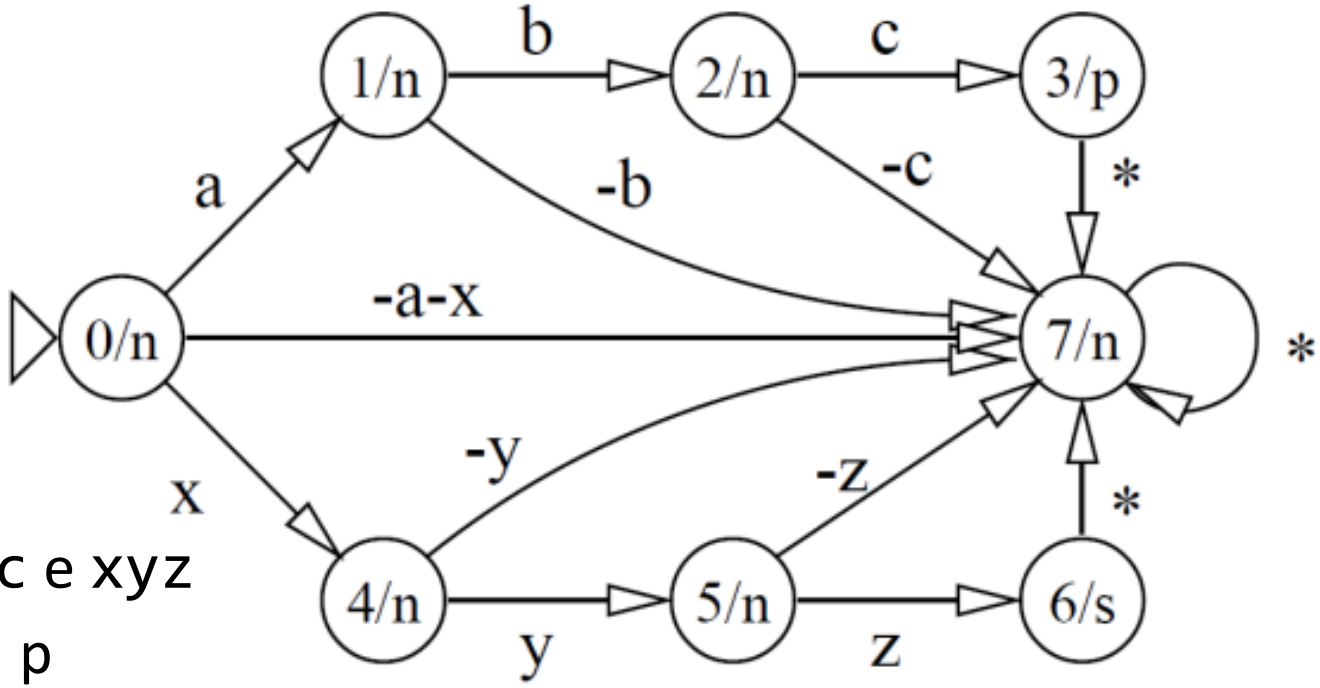
ma si possono mettere anche più uscite

sul disegno:

stato → stato/uscita

in ogni stato si mette la sua uscita

Esempio



riconosce abc e xyz

abc → uscita p

xyz → uscita s

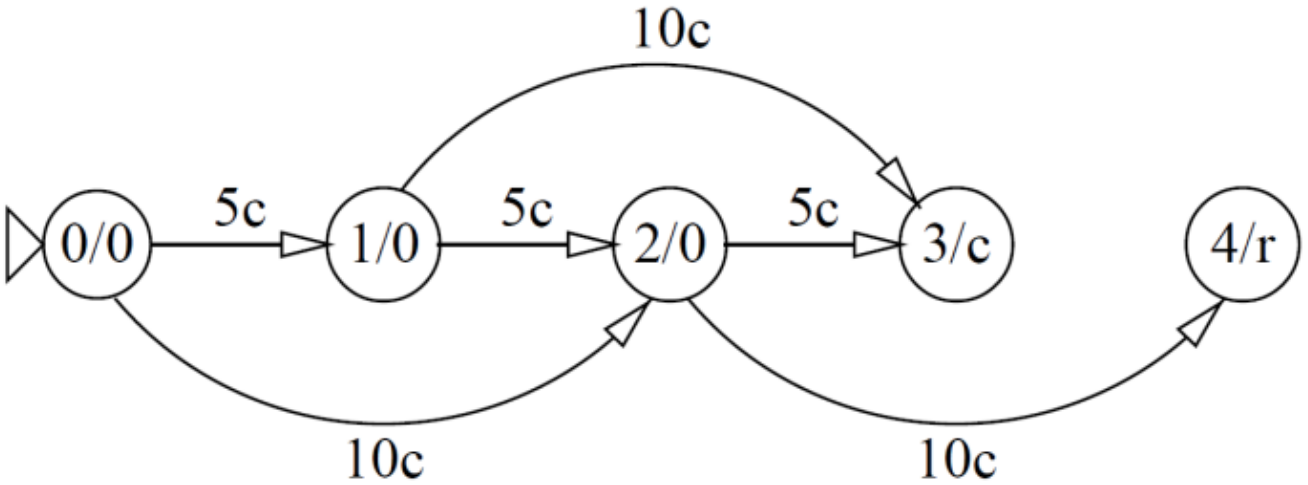
nessuna delle due: uscita n

non serve più mettere uno stato accettante

macchinetta del caffè con resto

uscite possibili:

- non abbastanza monete, niente caffè (0 nel disegno)
- monete esatte, caffè (c nel disegno)
- monete in eccesso, caffè + resto (r nel disegno)



manca: ricomincia dall'inizio dopo l'erogazione