Automi, espressioni regolari, grammatiche di tipo 3

Compilatori

UN COMPILATORE è un programma che traduce un programma scritto in un linguaggio ad alto livello (es. JAVA) in uno scritto in linguaggio macchina.

Il linguaggio macchina è il linguaggio originale del calcolatore sul quale il programma deve essere eseguito ed è letteralmente l'unico linguaggio per cui possiamo affermare che il calcolatore lo "comprende"

Agli inizi si programmava in binario e si scrivevano sequenze di bit (impresa titanica e piena di errori). Poco dopo furono sviluppati primitivi traduttori ("assembler") che permettevano al programmatore di scrivere istruzioni in uno specifico codice mnemonico anzichè mediante sequenze binarie (di solito un'istruzione assembler corrisponde a una singola istruzione in linguaggio macchina.

Linguaggi come JAVA, invece, sono noti come linguaggi ad alto livello ed hanno la proprietà che una loro singola istruzione corrisponde a più di un'istruzione in linguaggio macchina.

Linguaggi ad alto livello

Il vantaggio principale dei linguaggi ad alto livello è la produttività.

1) È stato stimato che il programmatore medio può produrre 10 linee di codice senza errori in una giornata di lavoro (il punto chiave è "senza errori": possiamo tutti scrivere enormi quantità di codice in minor tempo, ma il tempo addizionale richiesto per verificare e correggere riduce tale numero drasticamente).

2) Questo dato è essenzialmente indipendente dal linguaggio di programmazione utilizzato. Poiché una tipica istruzione in linguaggio ad alto livello può corrispondere a 10 istruzioni in linguaggio assemblatore, ne segue che approssimativamente possiamo essere 10 volte più produttivi se programmiamo in JAVA invece che in linguaggio assembler.

Linguaggi sorgente e codice oggetto

- linguaggio sorgente: il linguaggio ad alto livello che il compilatore riceve in ingresso; il programma in linguaggio sorgente che deve essere compilato è detto codice sorgente.
- linguaggio oggetto: il linguaggio in cui viene tradotto il programma; l'uscita del compilatore è il codice oggetto
- Di solito il linguaggio oggetto è un linguaggio macchina

Linguaggi sorgente e codice oggetto

- Abbiamo detto che il linguaggio oggetto è un linguaggio macchina ma non sempre è così
- Esempio: JAVA viene normalmente compilato nel linguaggio di una macchina virtuale, detto “byte code”.
- In tal caso, il codice oggetto va ulteriormente compilato o interpretato prima di ottenere il linguaggio macchina.
- Questo può sembrare un passo inutile ma il linguaggio byte code è relativamente facile da interpretare e l’utilizzo di una macchina virtuale consente di costruire compilatori per JAVA che siano indipendenti dalla piattaforma finale su cui il programma deve essere eseguito.

Fasi di un compilatore

Un compilatore è un programma molto complesso è realizzato mediante un numero separato di parti; di solito queste parti sono cinque e sono eseguite in sequenza

1. Analisi lessicale
2. Analisi sintattica
3. Generazione codice intermedio
4. Ottimizzazione
5. Generazione del codice oggetto

Le ultime tre fasi vanno oltre gli obiettivi del corso

Cosa faremo

1) Analisi lessicale: il compilatore scompone il codice sorgente in unità significanti dette token.

- Questo compito è relativamente semplice per la maggior parte dei moderni linguaggi di programmazione
- gli strumenti di cui faremo uso sono le espressioni regolari e gli automi a stati finiti.

2) Analisi sintattica: in questa fase, il compilatore determina la struttura del programma e delle singole istruzioni.

- Di questa fase ci occuperemo successivamente e vedremo che la costruzione di analizzatori sintattici utilizza le tecniche e i concetti sviluppati nella teoria dei linguaggi formali e, in particolare, sulle grammatiche generative libere da contesto (grammatiche di tipo 2)

Analizzatore lessicale

Il compito dell'analizzatore lessicale consiste nello scandire la sequenza del codice sorgente e scomporla in parti significanti, ovvero nei token

Esempio

- data l'istruzione JAVA if (x == y*(b - a)) x = 0;
- l'analizzatore lessicale deve essere in grado di isolare la parola chiave if, gli identificatori x, y, b, a, gli operatori ==, *, -, =, le parentesi, il letterale 0 ed il punto e virgola finale.

Analizzatore lessicale

Il compito dell'analizzatore lessicale consiste nello scandire la sequenza del codice sorgente e scomporla in parti significanti, ovvero nei token

Questo comporta l'esame del codice sorgente carattere per carattere; l'output è una sequenza di token

In questo scansione l'analizzatore lessicale può prendersi cura anche di altre cose come, ad esempio, la rimozione dei commenti, la conversione dei caratteri, la rimozione degli spazi bianchi

Analizzatore lessicale

Nell'istruzione JAVA if (x == y*(b - a)) x = 0;

- Durante l'analisi lessicale gli identificatori giocano tutti lo stesso ruolo: è sufficiente indicare che il prossimo oggetto nel codice sorgente è un identificatore (sarà chiaramente importante, in seguito, essere in grado di distinguere i vari identificatori).
- Analogamente, dal punto di vista sintattico, un letterale intero è equivalente ad un altro letterale intero:
- Infatti la struttura grammaticale dell'istruzione nel nostro esempio non cambierebbe se 0 fosse sostituito con 1 oppure con 1000 o al posto di x avessimo z.
- Così tutto quello che in qualche modo dobbiamo dire è di aver trovato un letterale intero (di nuovo, in seguito, dovremo distinguere tra i vari letterali interi, poiché essi sono funzionalmente differenti anche se sintatticamente equivalenti).

Analizzatore lessicale

L'istruzione if (x == y*(b - a)) x = 0;

Diviene if (id == id * (id - id)) id = int;

Trattiamo questa distinzione nel modo seguente: il tipo generico, passato all'analizzatore sintattico, è detto token

- id rappresenta il token identificatore, int il token costane intera, '==' token di confronto, '(' token parentesi ecc.

Le specifiche istanze del tipo generico sono dette lessemi.

- nel nostro esempio abbiamo quattro istanze (i lessemi x, y, b ed a) del token id (identificatore), e un'istanza (ovvero il lessema 0) del token int (letterale intero)

In altre parole un token è il nome di un insieme di lessemi che hanno lo stesso significato grammaticale per l'analizzatore sintattico.

Analizzatore lessicale

L'analizzatore lessicale deve isolare i token

In realtà, l'analizzatore lessicale ha anche un compito aggiuntivo: quando un identificatore viene trovato, deve dialogare con il programma gestore della tabella dei simboli.

- Se l'identificatore viene dichiarato, un nuovo elemento verrà creato, in corrispondenza del lessema, nella tabella stessa.
- Se l'identificatore viene invece usato, l'analizzatore deve verificare nella tabella dei simboli di verificare che esista un elemento per il lessema nella tabella.

I linguaggi regolari

- Sono i linguaggi generati da grammatiche di Chomsky di tipo 3.
- Sono usati nella fase di analisi lessicale
- Godono di varie interessanti proprietà. La più importante è:
- I linguaggi regolari
  - Sono definibili con grammatiche di tipo 3
  - Sono definibili con le espressioni regolari
  - Sono riconoscibili con automi a stati finiti
- Le tre definizioni danno la stessa classe di linguaggi

Automi a stati finiti

- Sono il tipo più semplice di macchina di calcolo (ideale) per riconoscere linguaggi
- Dispositivi che leggono la stringa di input da un nastro unidirezionale e la elaborano usando una memoria limitata
- Ad ogni passo: lettura di un carattere, spostamento della testina, aggiornamento dello stato della memoria.

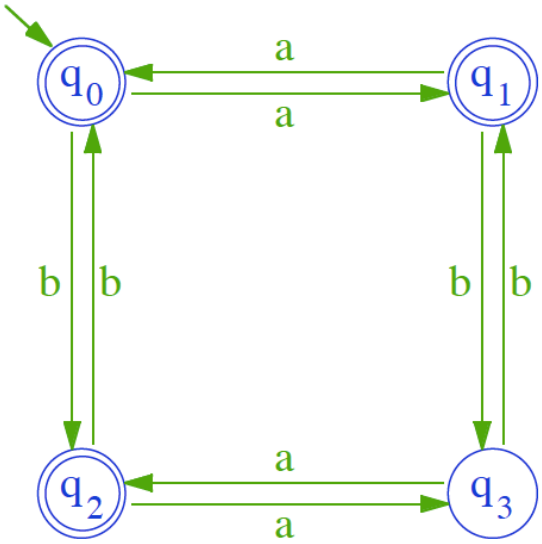
Automi a stati finiti deterministici

Def. Un Automa a stati finiti deterministico(nel seguito Automa) è definito da una quintupla $A=<\Sigma,Q,\delta,q0,F>$

- $\Sigma = \{\sigma1,...,\sigma n\}$ alfabeto di input
- $Q = \{q0,...,qm\}$ insieme finito non vuoto di stati
- $Q \supseteq F$ insieme di stati finali, $q0 \in Q$ stato iniziale
- funzione di transizione, $\delta: Q \times \Sigma \rightarrow Q$ funzione che determina lo stato successivo

La funzione di transizione di un automa può essere rappresentata mediante

- matrice (tabella) di transizione
- diagramma degli stati



Computazione eseguita da automi.

- Configurazione di A: coppia <q,x>
  con $q \in Q$ stato e $x \in \Sigma^*$ stringa da leggere in input
- Relazione di transizione di stato di un ASF:
  relazione binaria sulle configurazioni $|-$
  $\langle q,x \rangle |- \langle q',x' \rangle \Leftrightarrow x=ax' \wedge \delta(q,a)=q'$
- Configurazioni iniziale, finale e accettante di un ASF:
  <q,x> (ε denota "nessun carattere")
  iniziale se q = q0
  finale se x = ε (la stringa di input è finita)
  accettante se x = ε e $q \in F$

Linguaggio definito da un automa

Si introduce ε che rappresenta il carattere "nessun carattere" (input nullo)

Def. Funzione di transizione estesa alle stringhe

$\underline{\delta} : Q \times \Sigma^* \rightarrow Q$ (input una stringa)

$\underline{\delta}(q,\varepsilon) = q$ (con input nullo, rimani nello stato)

$\underline{\delta}(q,ax) = \delta(\underline{\delta}(q,a),x)$, con $x \in \Sigma^*$ e $a \in \Sigma$

Esempio

se y=abc abbiamo $\underline{\delta}(q,y) = \delta(\delta(\delta(q,a),b),c)$

Def. Linguaggio riconosciuto da un automa A:

$L(A) = \{ x \in \Sigma^* \mid \delta(q0,x) \in F \}$

Esempio

Dato il linguaggio $\{a^n b \mid n \geq 0\}$ generato da $S \rightarrow aS \mid b$,

l'automa che lo riconosce è l'automa

stato iniziale indicato con ⟹ stati finali con 

<{a,b}, {q0,q1,q2}, δ, q0, {q1}>

δ	a	b
q0	q0	q1
q1	q2	q2
q2	q2	q2

input

Nuovo stato

Stato attuale

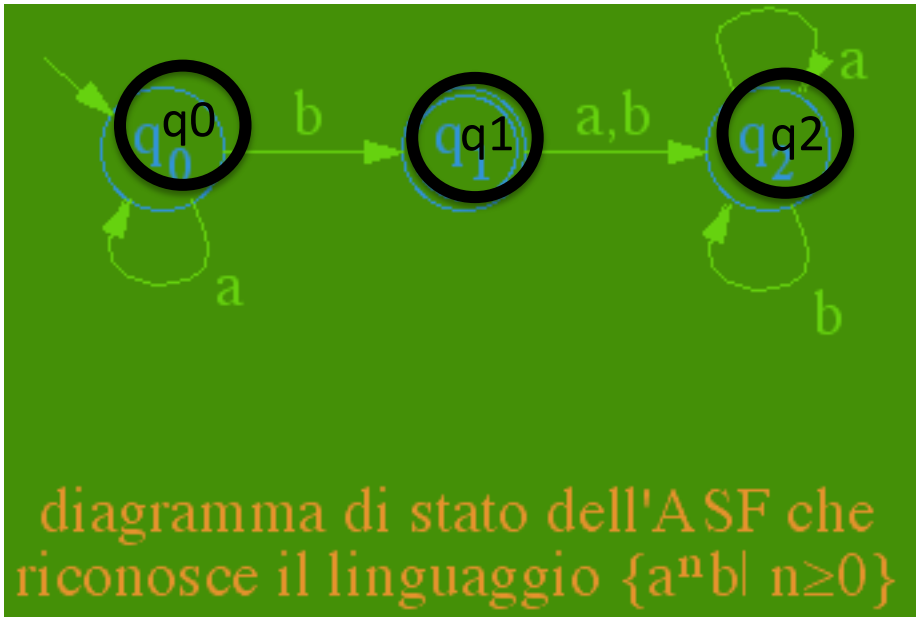


diagramma di stato dell'ASF che riconosce il linguaggio $\{a^n b \mid n \geq 0\}$

Esempio

Esempio. Automa che riconosce il linguaggio delle parole che contengono un numero pari di a o un numero pari di b (assumo 0 pari)

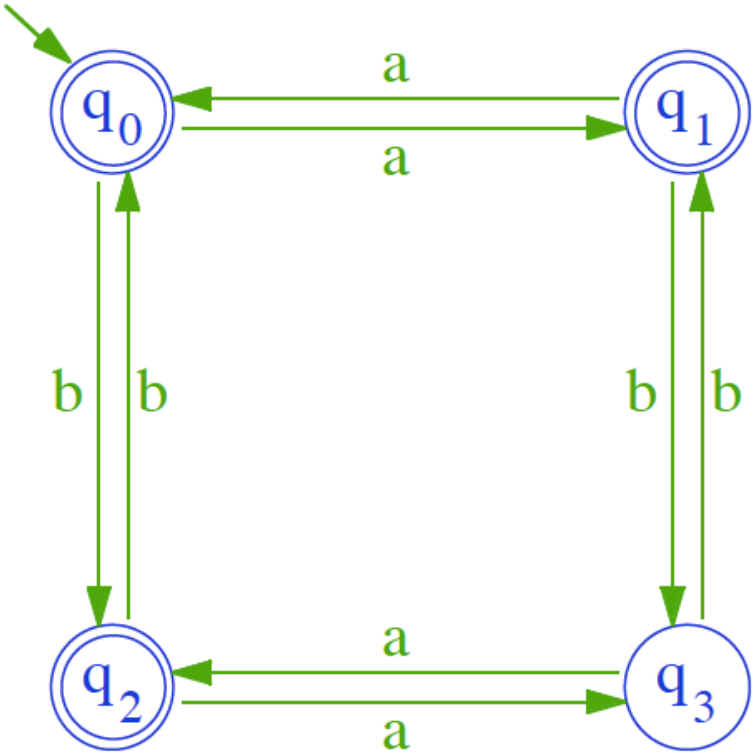
Σ={a,b}

Q = {q0, q1, q2, q3}

F = {q0, q1, q2}

Funzione di stato δ

δ	a	b
q0	q1	q2
q1	q0	q3
q2	q3	q0
q3	q2	q1



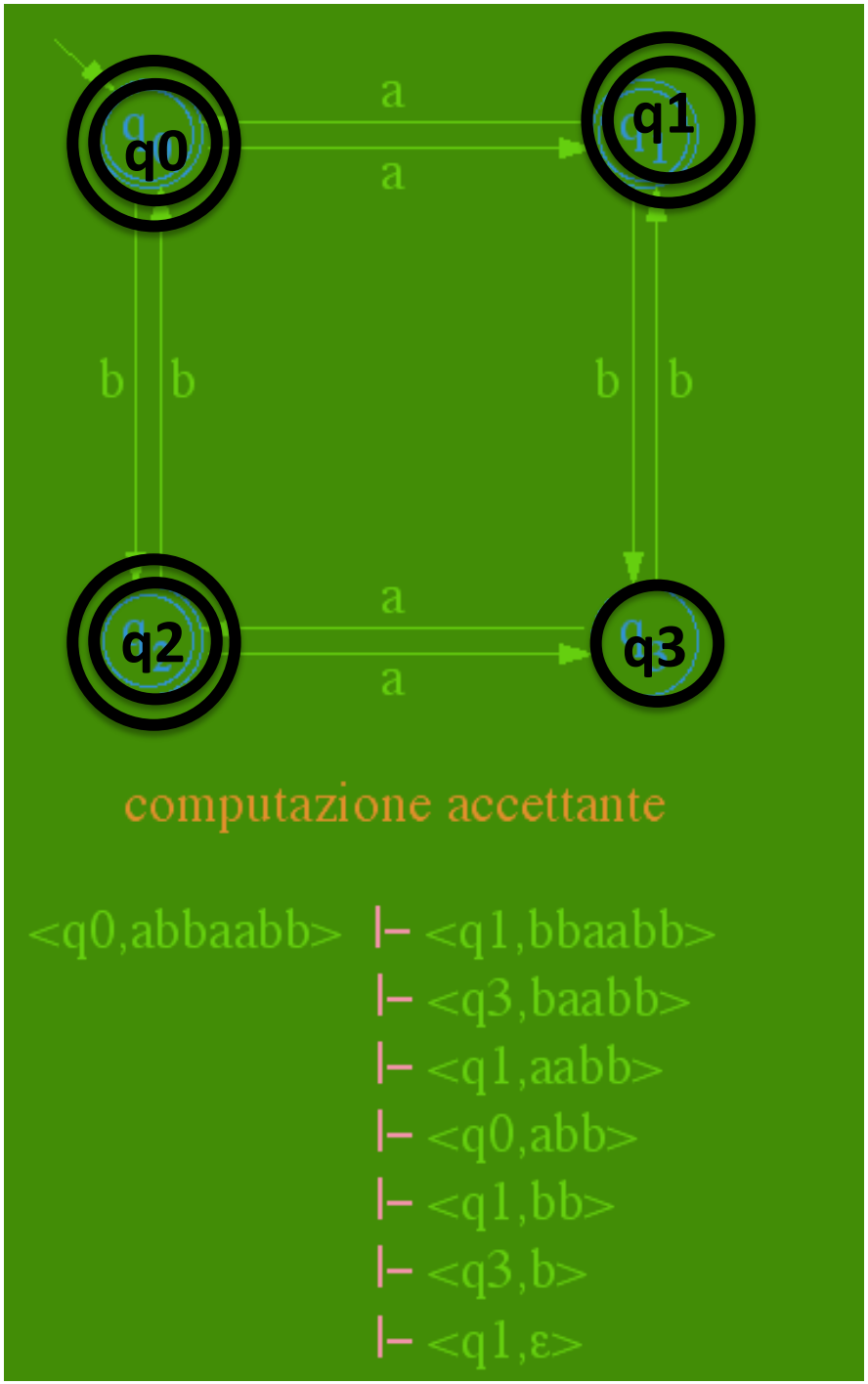
Esempio

L'automa che riconosce le stringhe con un numero pari di a o di b accetta la stringa abbaabb

δ	a	b
q0	q1	q2
q1	q0	q3
q2	q3	q0
q3	q2	q1

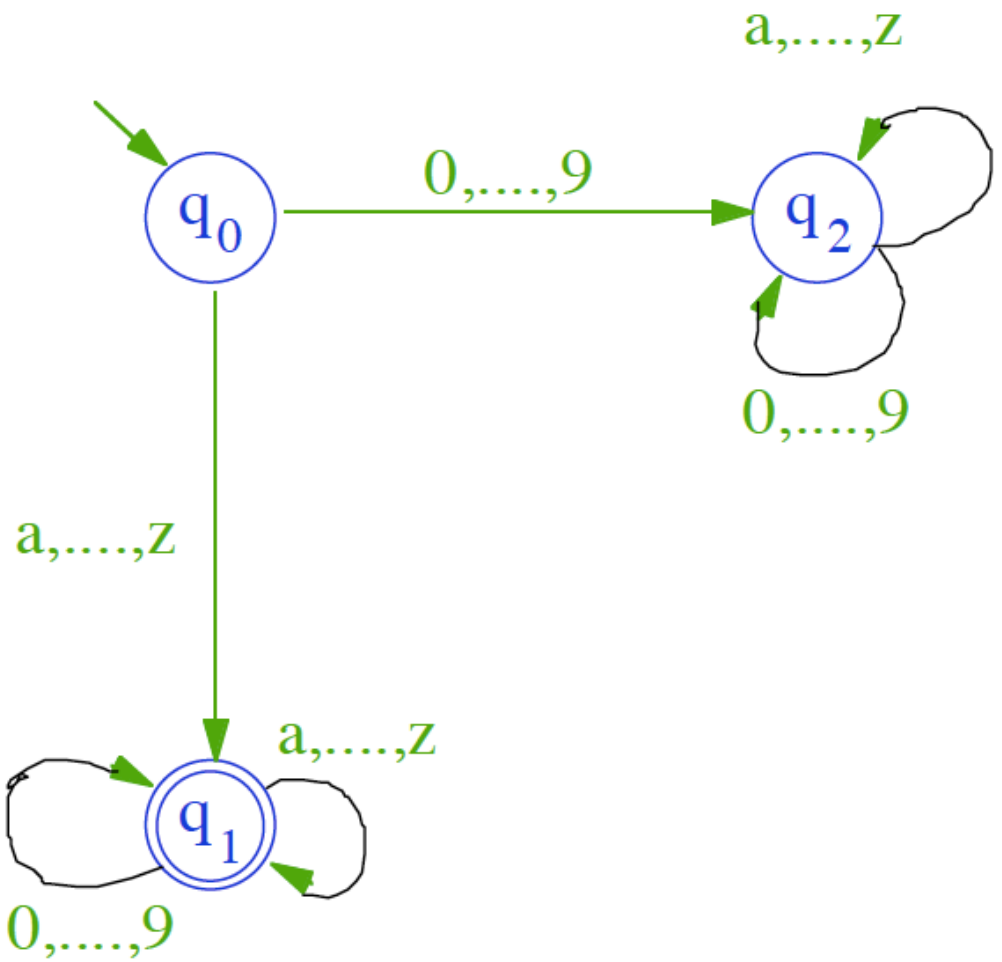
la computazione accettante

δ(q0,abbaabb)= δ(q1,bbaabb)= …
δ(q3,b)= δ(q1,ε)



Automa che riconosce gli identificatori

Nella grande maggioranza dei linguaggi un identificatore è definito come una sequenza di caratteri alfanumerici che inizia con un carattere alfabetico



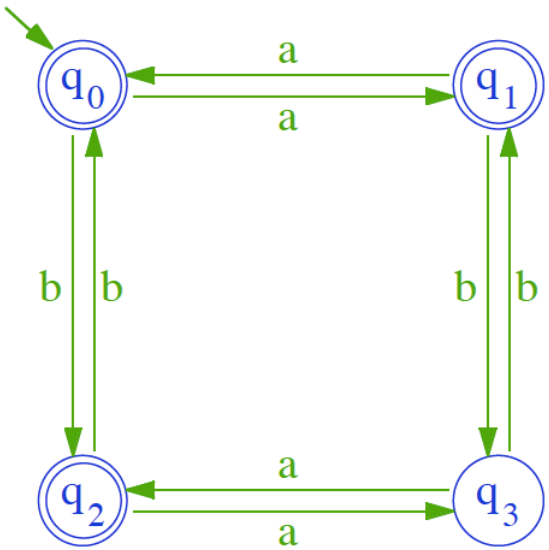
Automi a stati finiti nondeterministici

Ricordiamo la definizione di automa deterministico

Def. Un Automa a stati finiti deterministico (nel seguito Automa) è definito da una quintupla $A=<\Sigma,Q,\delta,q0,F>$

- $\Sigma = \{\sigma1,...,\sigma n\}$ alfabeto di input
- $Q = \{q0,...,qm\}$ insieme finito non vuoto di stati
- $Q \supseteq F$ insieme di stati finali, $q0 \in Q$ stato iniziale
- funzione di transizione, $\delta: Q \times \Sigma \rightarrow Q$

  che determina lo stato successivo



Automi a stati finiti nondeterministici

Modifica per definire un automa nondeterministico

Def. Un Automa a stati finiti nondeterministico è definito da una quintupla $A=<\Sigma,Q,\delta,q0,F>$

- $\Sigma = \{\sigma1,...,\sigma n\}$ alfabeto di input
- $Q = \{q0,...,qm\}$ insieme finito non vuoto di stati
- $Q \supseteq F$ insieme di stati finali, $q0 \in Q$ stato iniziale
- funzione di transizione, ~~$\delta: Q \times \Sigma \rightarrow Q$~~

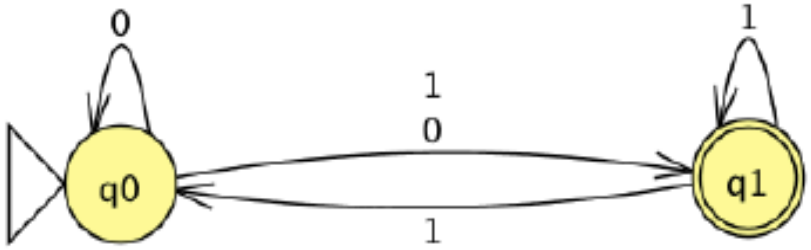
  $\delta: Q \times \Sigma \rightarrow P(Q)$

dove P(Q) rappresenta l'insieme delle parti di Q (l'insieme dei sottoinsiemi di Q)

Esempio automa nondeterministico

Un automa con due stati q0 e q1
q0 stato iniziale
q1 stato finale
La funzione di transizione è data dalla tabella

stato	simbolo	insieme di stati
q0	0	{q0, q1}
q0	1	{q1}
q1	1	{q0, q1}



Automi a stati finiti nondeterministici

Def. Un Automa a stati finiti nondeterministico è definito da una quintupla $A=<\Sigma,Q,\delta,q0,F>$

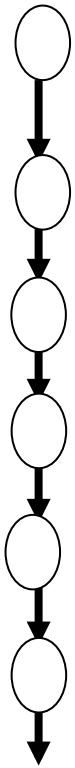
- $\Sigma = \{\sigma1,...,\sigma n\}$ alfabeto di input
- $Q = \{q0,...,qm\}$ insieme finito non vuoto di stati
- $Q \supseteq F$ insieme di stati finali, $q0 \in Q$ stato iniziale
- funzione di transizione, ~~$\delta: Q \times \Sigma \rightarrow Q$~~

  $\delta: Q \times \Sigma \rightarrow P(Q)$ P(Q) l'insieme delle parti di Q

Il non determinismo è esteso includendo la possibilità di ε-transizioni, ovvero transizioni che avvengono senza leggere alcun simbolo di input: l'automa può cambiare stato anche senza leggere un carattere in ingresso

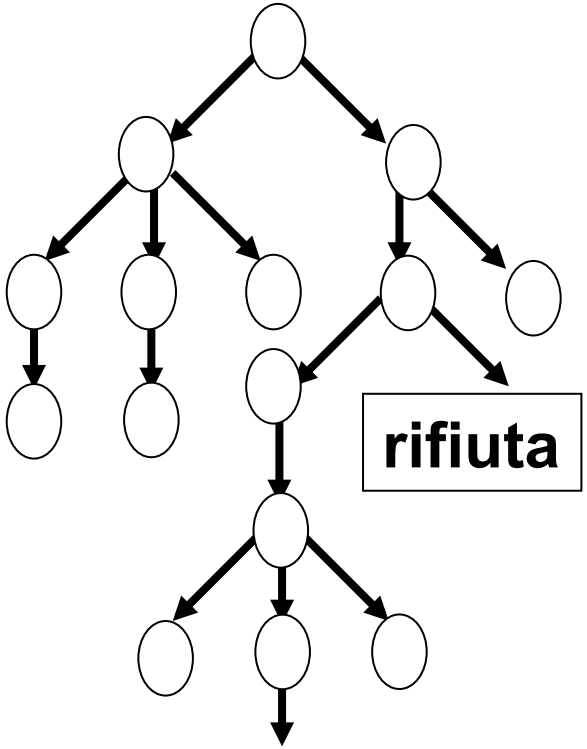
Computazioni deterministiche e non determ.

Computazione Deterministica



accetta o rifiuta

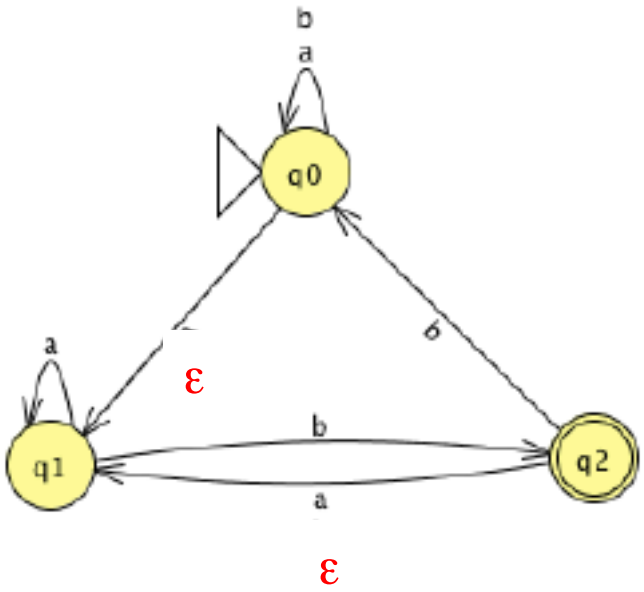
Computazione Non Deterministica



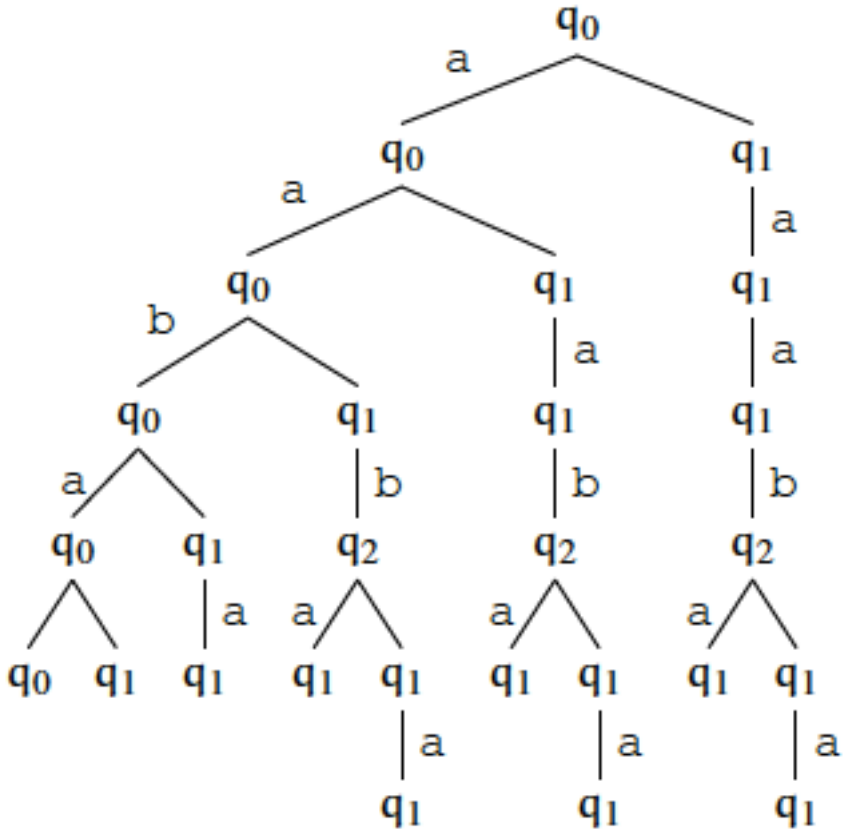
Nel caso non-deterministico l'automa accetta se esiste uno stato che accetta (anche uno solo)

Esempio automa nondeterministico

Un automa con stati q0, q1, q2 con due ε-transizione (da qo a q1 e da q2 a q1)



Albero computazioni con input aaba (automa rifiuta la stringa)



Automi determ. e automi nondeterm.

Automi deterministici e automi non deterministici sono equivalenti?

- Abbiamo visto che una macchina di Turing non deterministica può essere simulata da una deterministica, facendo uso della tecnica di visita in ampiezza dell'albero delle computazioni.
- Chiaramente, tale tecnica non è realizzabile mediante un automa a stati finiti (non ha un nastro su cui scrivere e non può tornare indietro sul nastro di input
- Con una tecnica diversa possiamo dimostrare
- Teorema Sia T un automa a stati finiti non deterministico. Allora, esiste un automa a stati finiti deterministico T' equivalente a T.

Teorema Sia T un automa a stati finiti non determinist.; esiste un automa a stati finiti T' equivalente a T.

- T' ha lo lo stesso alfabeto Σ di T; l'insieme degli stati è diverso
- La dimostrazione procede definendo uno dopo l'altro gli stati di T' sulla base degli stati già definiti e delle transizioni di T
- ogni stato di T' denota un sottoinsieme degli strati di T;
- lo stato iniziale di T' è lo stato {q0} (q0 stato iniziale di T)
- Gli stati finali di T' sono quegli insiemi di stati che includono almeno uno stato finale di T

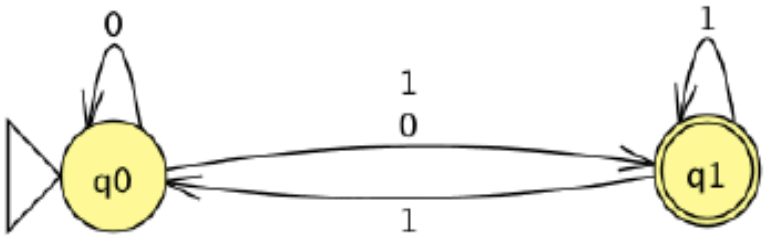
…prova (continua)

ogni stato di T' denota un sottoinsieme degli stati di T; lo stato iniziale di T' è lo stato {q0}; la costruzione delle transizioni e degli altri stati di T' procede così:

- Sia S = {s1, . . . ,sk} uno stato di T' per cui non è ancora stata definita la transizione corrispondente a un simbolo x di Σ e, per ogni i, i = 1, . . . ,k, sia N(si,x) l'insieme degli stati raggiungibili da si leggendo x (nota: S potrebbe anche essere l'insieme vuoto). Definiamo allora S= S1 U S2…U Sk (unione tutti i Si)
- Introduciamo lo stato S di T' (se non già presente)
- introduciamo la transizione di T' dallo stato Q allo stato S leggendo il simbolo x

Esempio Consideriamo l'automa con due stati q0 e q1
q0 stato iniz. q1 stato fin. e funzione di transizione data in tabella

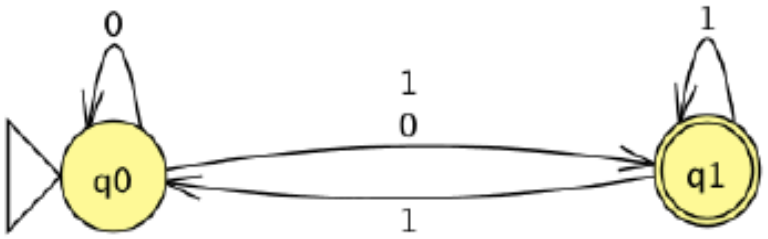
stato	simbolo	insieme di stati
q0	0	{q0, q1}
q0	1	{q1}
q1	1	{q0, q1}



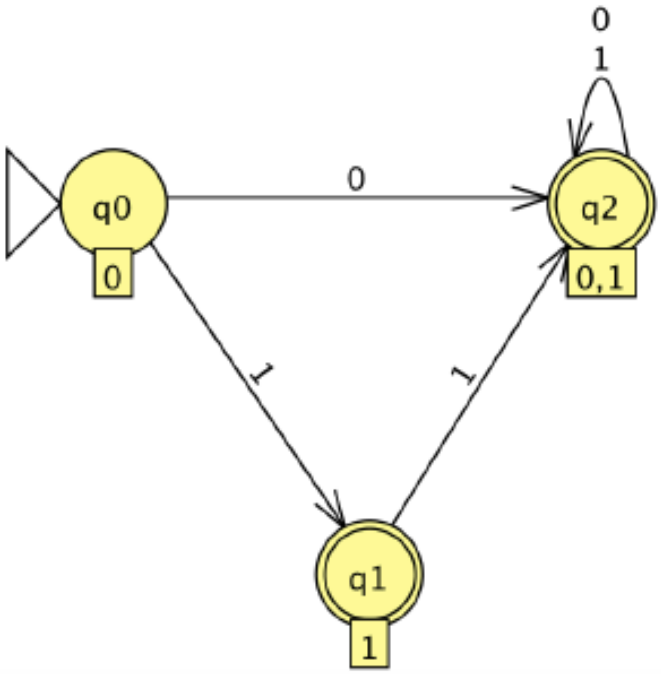
- {q0} è stato iniziale di T'
- Abbiamo che δ(q0,0) ={q0,q1} e δ(q0,1) = {q1} . Questi due stati non sono ancora stati generati: li aggiungiamo all'insieme degli stati di T' e definiamo una transizione verso di essi a partire dallo stato {q0} leggendo 0 e 1
- Per definire la transizione a partire da {q0,q1} leggendo 0 e otteniamo δ(q0,0) = {q0,q1} e δ(q1,0) = ∅ (ins. vuoto): l'unione di questi due insiemi è uguale a {q0,q1}, che già esiste; quindi definiamo la transizione da {q0;q1} a se stesso leggendo 0
- In modo analogo possiamo calcolare le transizioni a partire da {q1} e non creiamo nuovi stati di T'

Esempio Consideriamo l'automa con due stati q0 e q1
q0 stato iniz. q1 stato fin. e funzione di transizione data in tabella

stato	simbolo	insieme di stati
q0	0	{q0, q1}
q0	1	{q1}
q1	1	{q0, q1}



- Si continua fino a quando non si generano nuovi stati.
- Nel nostro caso abbiamo finito
  e gli stati di T' sono
  Q0= {q0}, Q1={q0,q1}, Q2{q1}
- L'automa nondeterministico è mostrato
  in figura a destra



Es. input 011 andiamo da Q0 a Q2 e poi rimaniamo in Q2 (accetta)
Input 1 andiamo da Q0 a Q1 (accetta)

Automi e linguaggi regolari

Teorema Un linguaggio L è regolare (tipo3) se e solo se esiste un automa a stati finiti che decide L.

1) L regolare → esiste un automa A

- Data grammatica G che genera L crea uno stato di A per ogni simbolo nonterminale di G
- Per ogni produzione del tipo X→ aY di G, A avrà una transizione dallo stato corrispondente a X allo stato corrispondente a Y leggendo il simbolo a

DA **X→ aY** **OTTENIAMO**

Automi e linguaggi regolari

Teorema Un linguaggio L è regolare (tipo3) se e solo se esiste un automa a stati finiti che decide L

2) automa A decide L → L regolare

Dato automa che decide L definisci grammatica G con un simbolo nonterminale per ogni stato di A

Per ogni transizione che dallo stato X fa passare l'automa nello stato Y leggendo il simbolo a, G avrà una produzione X→ aY

DA 

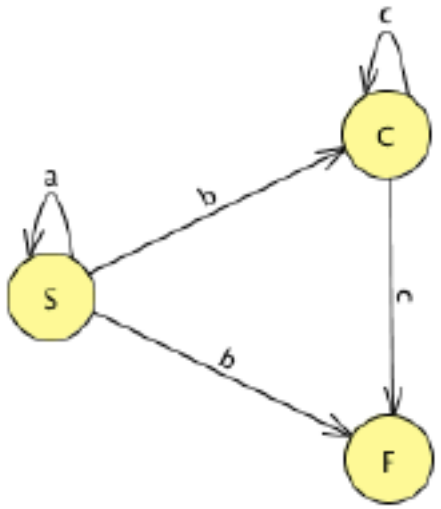
OTTENIAMO **X→ aY**

Automi e linguaggi regolari

Teorema Un linguaggio L è regolare (tipo3) se e solo se esiste un automa a stati finiti che decide L.

- La prova fornisce un automa nondeterministico; sappiamo che per ogni automa nondeterministico ne esiste uno deterministico equivalente

$S \to aS \quad S \to bC \quad S \to b$
$C \to cC \quad C \to c$



Conclusioni

Gli automi a stati finiti consentono di rappresentare in generale sistemi di transizione in un insieme finito di stati

Esempi: un ascensore,un distributore automatico di bevande

- Automi deterministici e automi nondeterministci sono equivalenti
- I linguaggi definiti dagli automi sono i linguaggi regolari
- Gli automi non possono scrivere su una memoria (come ad es. le macchine di Turing); per questo i linguaggi regolari sono troppo semplici per definire le grammatiche dei linguaggi di programmazione

Espressioni regolari

Dato un alfabeto Σ un linguaggio L è un sottoinsieme di Σ^*

Quindi un linguaggio finito può essere definito elencandone le stringhe

Esempio

Se Σ= {a,b} un possibile linguaggio su Σ è L= {a,aa,bb}

Se Σ= {a,b,c,d,e,f,g} un possibile linguaggio su Σ è L= {cade, daga, fa, bacca}

Questo approccio non permette di definire linguaggi infiniti.

Le grammatiche, gli automi e le espressioni regolari permettono di definire linguaggi infiniti.

Le espressioni regolari usano un approccio algebrico

Espressioni regolari

Ricordiamo le operazioni elementari su linguaggi

Nel seguito ε (λ) rappresenta la stringa vuota

Dati due linguaggi L1 e L2 su Σ^* definiamo

- Unione $L1 \cup L2 = \{x \in \Sigma^* \mid x \in L1 \vee x \in L2\}$

  $L1 \cup \{\varepsilon\} = L1$

Nota si usa anche simbolo + e il termine selezione (si seleziona un elemento fra I due linguaggi)

- Intersezione $L1 \wedge L2 = \{x \in \Sigma^* \mid x \in L1 \wedge x \in L2\}$

  $L1 \wedge \{\varepsilon\} = \{\varepsilon\}$

- Complementazione

  $L1c = \{x \in \Sigma^* \mid x \notin L1\}$

Espressioni regolari

Dati due linguaggi L1 e L2 su Σ^* definiamo

- Concatenazione (prodotto) di linguaggi

  $L1 \cdot L2 = \{x \in \Sigma^* \mid$ esistono x1 ed x2 tali che $x1 \in L1 \wedge$ $x2 \in L2,\ x=x1 \cdot x2\}$

  $L \cdot \{\varepsilon\} = \{\varepsilon\} \cdot L = L \qquad L \cdot \lambda = \lambda \cdot L = \lambda$

Spesso invece di $L1 \cdot L2$ scriviamo L1 L2

NOTA BENE L'operazione di concatenazione è associativa ma non è commutativa (come la concatenaz. di stringhe)

Espressioni regolari

Dati due linguaggi L1 e L2 su Σ^* definiamo

- Potenza di un linguaggio L

  $L^h = L\ L^{h-1}$, $h \geq 1$

  $L^0 = \{\varepsilon\}$ per convenzione

- Iterazione (di Kleene) di un linguaggio L: $L^* = \bigcup_{h=0}^{\infty} L^h$

NOTA BENE $\varepsilon \in L^*$ (per definizione)

Se si vuole indicare il linguaggio L^* escludendo la stringa vuota si usa il simbolo L^+

$L^+ = \bigcup_{h=1}^{\infty} L^h$ quindi $L^* = L^+ \cup \{\varepsilon\}$

Esempi

- L1= {Auguri., Congratulazioni., Condoglianze.},
  L2 = {Giorgio, Lucia}

  L1 L2 = {Auguri. Giorgio, Auguri. Lucia,
  Condoglianze. Giorgio, Condoglianze. Lucia
  Congratulazioni.Giorgio, Congratulazioni. Lucia}
- L={ab, aab}

  L*={ε, ab, aab, abab, aabaab, ababab, ecc.}
- L={a, b}, L* contiene tutte le stringhe definite sui due simboli a e b, inclusa la stringa vuota

Espressioni regolari

L'insieme delle espressioni regolari su di un alfabeto Σ è così definito induttivamente

- Ogni carattere in Σ è un'espressione regolare
- ε è un'espressione regolare (stringa vuota)
- Se R ed S sono due espressioni regolari, allora
  - La concatenazione (o prodotto) R S è un'espressione regolare
  - L'unione (o selezione) R+S è un'espressione regolare.
  - La chiusura di Kleene R* è un'espressione regolare.
- Solo le espressioni formate da queste regole sono regolari.

Espressioni regolari

Un'espressione regolare R genera un linguaggio L(R) definito in modo induttivo nel modo seguente.

- Se R = a ($\in \Sigma$) , allora L(R) = {a}
- Se R = ε , allora L(R) = $\{\varepsilon\}$
- Se R = S1 S2, allora

  L(R) = L(S1 S2)= {xy : x$\in$ L(S1) e y$\in$L(S2)}
- Se R = S1+S2, allora L(R) = L(S1+ S2)= L(S1) U L(S2)
- Se R = S*, allora

  L(R) = {$x^1x^2 \ldots x^n$: n $\geq$ 0 e $x^i \in$ L(S)}

Espressioni regolari: proprietà

Se R, S e T sono tre espressioni regolari, allora le seguenti affermazioni sono vere.

- Associatività: R(ST) è equivalente a (RS)T.
- Associatività R+(S+T) è equivalente a (R+S)+T.
- Commutatività R+S è equivalente a S+R.
- Distributività R(S+T) è equivalente a RS+RT.
- Identità R $\{\varepsilon\}$ e $\{\varepsilon\}$R sono equivalenti a R

La concatenazione non è commutativa in quanto, in generale, L(RS) ≠ L(SR)

Nel valutare un'espressione regolare assumiamo che la chiusura di Kleene abbia priorità maggiore mentre la selezione abbia priorità minore; le parentesi sono usate per annullare le priorità nel modo usuale

Espressioni regolari: proprietà

Se R, S e T sono tre espressioni regolari, allora le seguenti affermazioni sono vere.

- Associatività: R(ST) è equivalente a (RS)T
- Associatività R+(S+T) è equivalente a (R+S)+T
- Commutatività R+S è equivalente a S+R
- Distributività R(S+T) è equivalente a RS+RT.
- Identità R $\{\varepsilon\}$ e $\{\varepsilon\}$R sono equivalenti a R

La concatenazione non è commutativa in quanto, in generale, L(RS) ≠ L(SR)

Nel valutare un'espressione regolare assumiamo che la chiusura di Kleene abbia priorità maggiore mentre la selezione abbia priorità minore; le parentesi sono usate per annullare le priorità nel modo usuale

Espressioni regolari: proprietà

Nel valutare un'espressione regolare assumiamo che la chiusura di Kleene abbia priorità maggiore mentre la selezione abbia priorità minore

Esempi:

L(ab*)= {a, ab, abb, abbb,…} infatti * ha priorità su concatenazione; quindi L(ab*) non è

{ab, aabb, aaabbb}

L(ab+ca)={ab,ca} la concatenazione ha priorità maggiore di +; quindi L(ab+ca) non è {aba, aca}

Le parentesi sono usate per annullare le priorità nel modo usuale

Esempi

- ab*(bab+aba) rappresenta il linguaggio costituito da tutte le stringhe che cominciano con a, proseguono con un numero arbitrario (anche nullo) di b e terminano con la stringa bab o con aba
- (1*01*01*)* rappresenta un'espressione regolare che genera l'insieme di tutte le sequenze di 0 ed 1 che contengono un numero di 0 pari (anche nessun 0).

  Esercizio: mostrare che (1*001*)* e (01*01*)* definiscono linguaggi diversi da (1*01*01*)*

Esempi: riconoscimento dei token

Le espressioni regolari sono abbastanza potenti da poter essere usate per definire i token dell'analisi lessicale

Abbiamo visto, infatti, che un token può essere visto come un linguaggio che include i suoi lessemi (cioè le possibili istanze del token)

(0+(1+2+3+4+5+6+7+8+9)(0+1+2+3+4+5+6+7+8+9)*)(ε+l+L)

rappresenta il token letterale intero decimale in JAVA; Infatti, tale token è definito come un numerale decimale eventualmente seguito dal suffisso l oppure L allo scopo di indicare se la rappresentazione deve essere a 64 bit.

Un numerale decimale può essere uno 0 oppure una cifra da 1 a 9 eventualmente seguita da una o più cifre da 0 a 9; in questo modo non accettiamo 007 come numero decimale intero

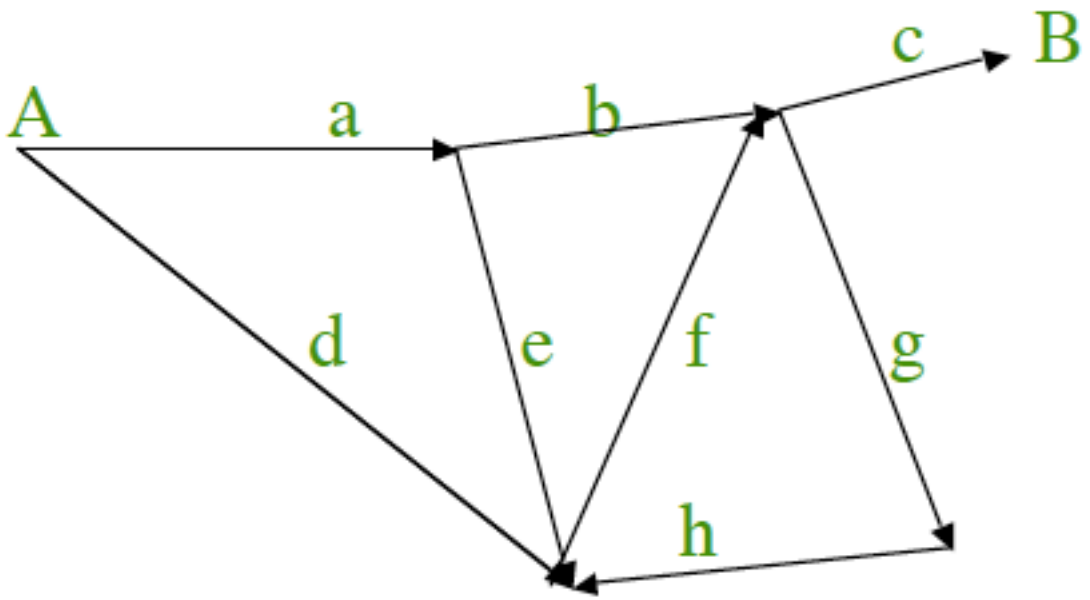
Esempi

Il seguente esempio ci permette di capire la struttura e l'uso delle espressioni regolari.

Consideriamo la rete stradale in figura in cui i nodi sono stati (rappresenta lo stato in cui ci troviamo nella rete) e gli archi il passaggio da uno stato all'altro

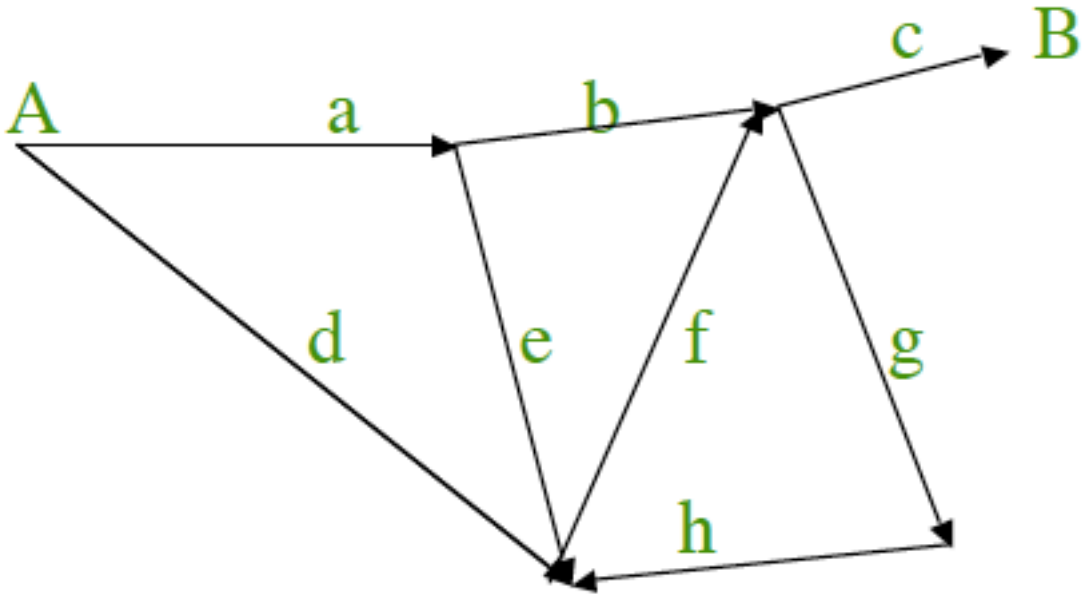
Supponiamo che A sia stato iniziale e B finale e ogni arco rappresenti un cambio di stato

Quindi abc rappresenta il passaggio (o il cammino) da A a B



Esempi

Sia data la rete stradale in figura.
abc rappresenta un cammino da A a B
altri possibili cammini sono dfc, aefc, abghf



(ab + aef + df)(ghf)*c è l'epressione regolare che rappresenta tutti i percorsi da A a B (inclusi i cicli)

Da espressioni regolari a automi stati finiti

Teorema Per ogni espressione regolare R esiste un automa a stati finiti non deterministico T tale che L(R) = L(T).

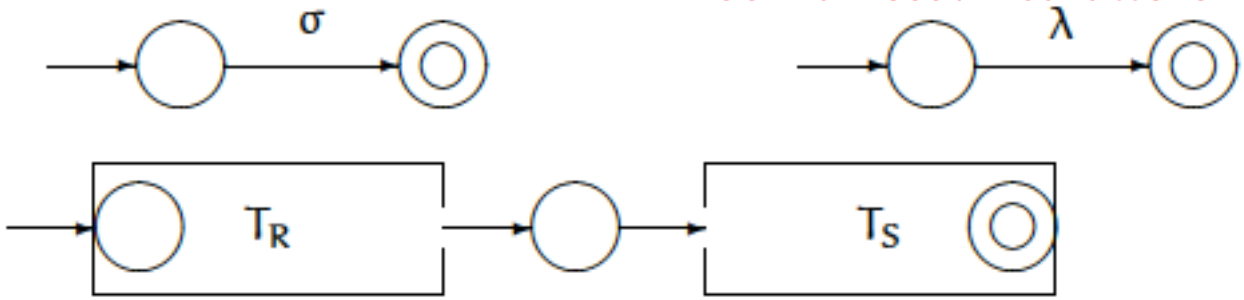
La costruzione si basa sulla definizione induttiva delle espressioni regolari fornendo una macchina oppure un'interconnessione di macchine (non deterministiche) corrispondente ad ogni passo della definizione (concatenazione, unione, chiusura transitivia)

La costruzione, che tra l'altro assicura che l'automa ottenuto avrà un solo stato finale diverso dallo stato iniziale.

NOTA BENE Nel seguito le figure utilizzano il simbolo λ al posto di ε per denotare la stringa vuota

Passo elementare e concatenazione

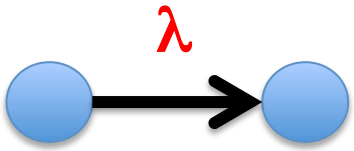
transizione di cambia stato
senza nessun carattere



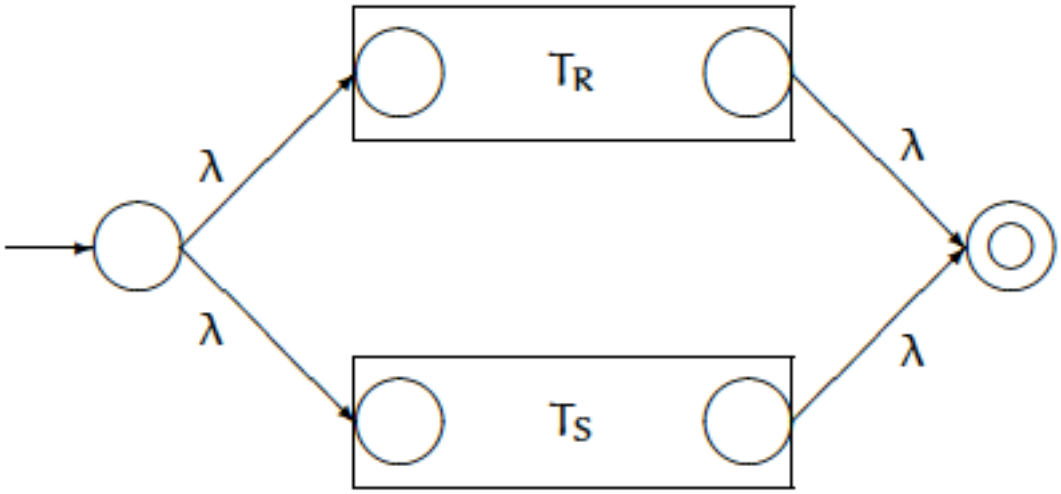
- La macchina in alto a sinistra accetta il carattere σ
- Quella in alto a destra accetta λ (ε, stringa vuota)
- Date due espressioni regolari R e S, la macchina mostrata nella seconda riga accetta L(RS) dove TR e TS denotano le due macchine che decidono, rispettivamente, L(R) ed L(S) e lo stato finale di TR `e stato fuso con lo stato iniziale di TS in un unico stato

la figure utilizza il simbolo λ al posto di ε per denotare la stringa vuota

Unione (+)

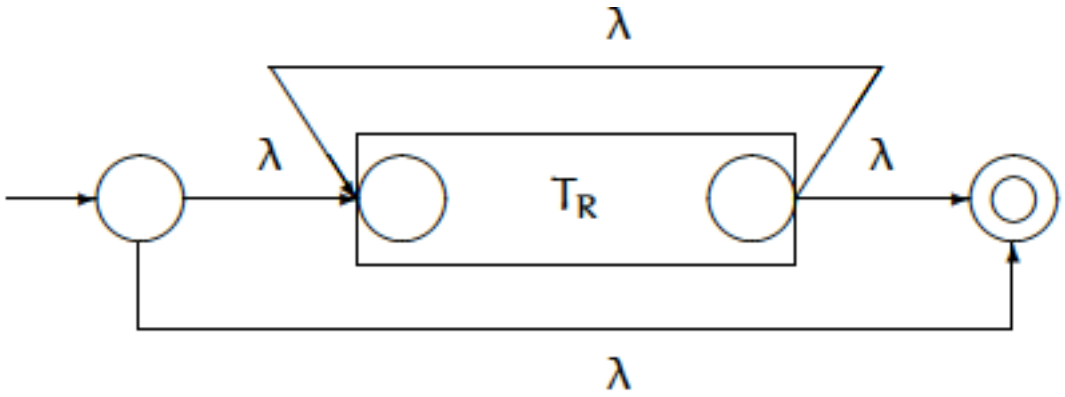


Rappresenta una transizione di cambia stato senza nessun carattere



- Date due espressioni regolari R e S, la macchina mostrata accetta L(R + S) dove TR e TS denotano le due macchine che decidono, rispettivamente, L(R) ed L(S)
- un nuovo stato iniziale è stato creato, due l-transizioni da questo nuovo stato agli stati iniziali di TR ed TS sono state aggiunte
- un nuovo stato finale è stato creato e due -transizioni dagli stati finali di TR ed TS a questo nuovo stato sono state aggiunte.

Chiusura tansitiva



- Data un'espressione regolare R, la macchina accetta L(R*) dove TR denota una macchina che riconosce L(R)
- un nuovo stato iniziale ed un nuovo stato finale sono stati creati,
- due –transizioni dal nuovo stato iniziale al nuovo stato finale e allo stato iniziale di TR sono state aggiunte e
- due -transizioni dallo stato finale di TR allo stato iniziale di TR e alnuovo stato finale sono state aggiunte.

Esempio

espressione regolare per letterale intero decimale

(0+{1,2,3,4,5,6,7,8,9}{0,1,2,3,4,5,6,7,8,9}*)(ε+l+L)

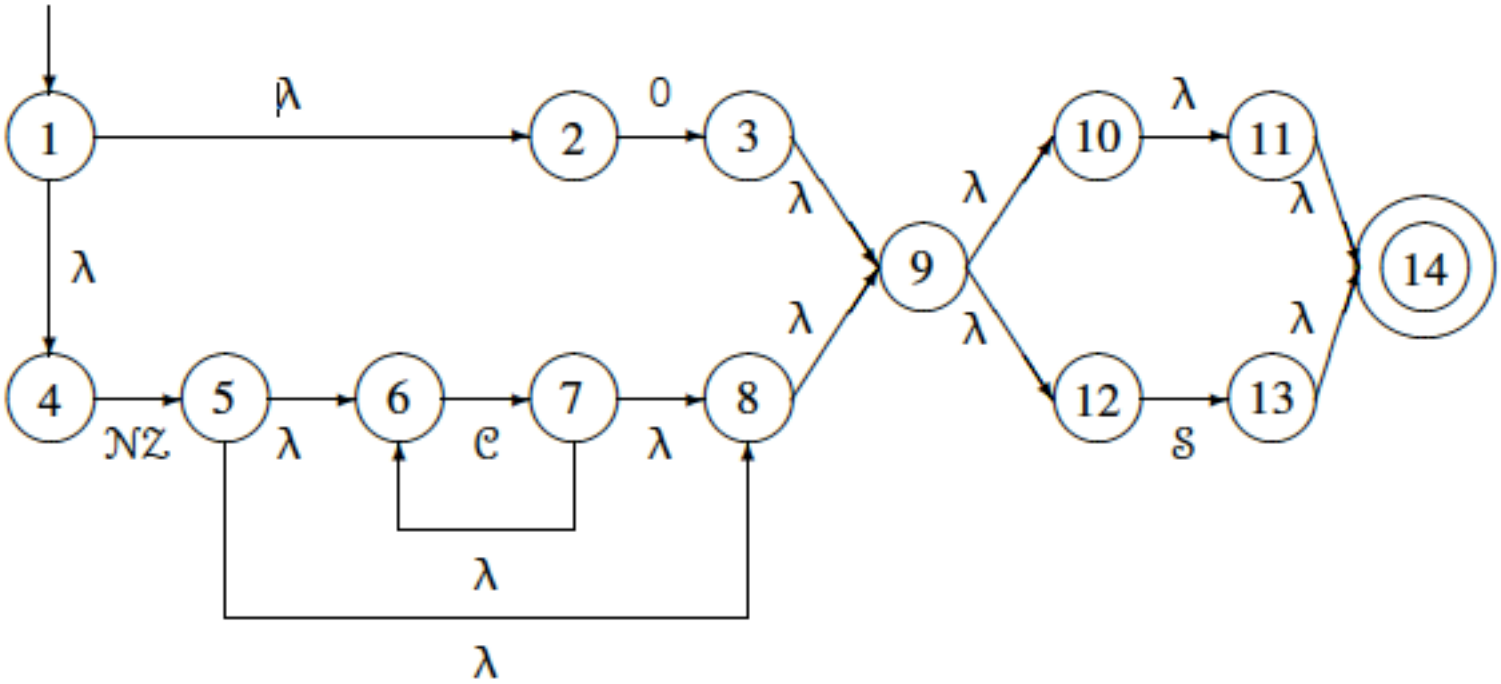
Per semplicità scriviamo l'espressione come (0+NZ C*)(S)

Dove *C* indica l'insieme delle cifre da 0 a 9

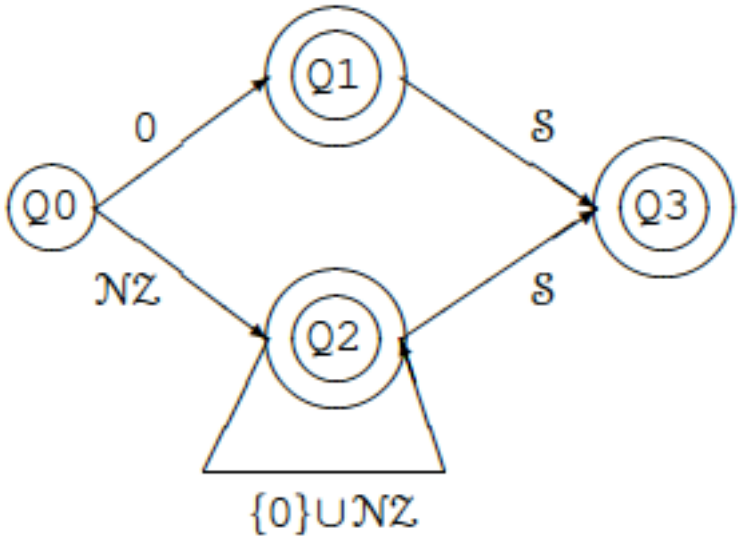
NZ l'insieme delle cifre da 1 a 9

S l'insieme {l, L}

λ denota una transizione (nondeterministica) senza input



Esempio



L'automa precedente è nondeterministico
Operando trasformazione
Automa nondeterministico → deterministico e semplificando si ottiene l'automa in figura

Equivalenze

Abbiamo visto che

1. Per ogni grammatica tipo 3 (regolare) G esiste un automa che accetta le stringhe definite da G
2. Per ogni automa a stati finiti A esiste una grammatica regolare che definisce il linguaggio accettato da A
3. Per ogni espressione regolare E esiste un automa che accetta le stringhe definite da E

Possiamo anche dimostrare (non faremo la prova solo il metodo ed esempi) che

4. Per ogni grammatica regolare G esiste una espressione regolare che definisce lo stesso linguaggio generato da G

Conclusione: Automi stati finiti, grammatiche tipo 3 e espressioni regolari definiscono la stessa classe di linguaggi

Equivalenze

Abbi

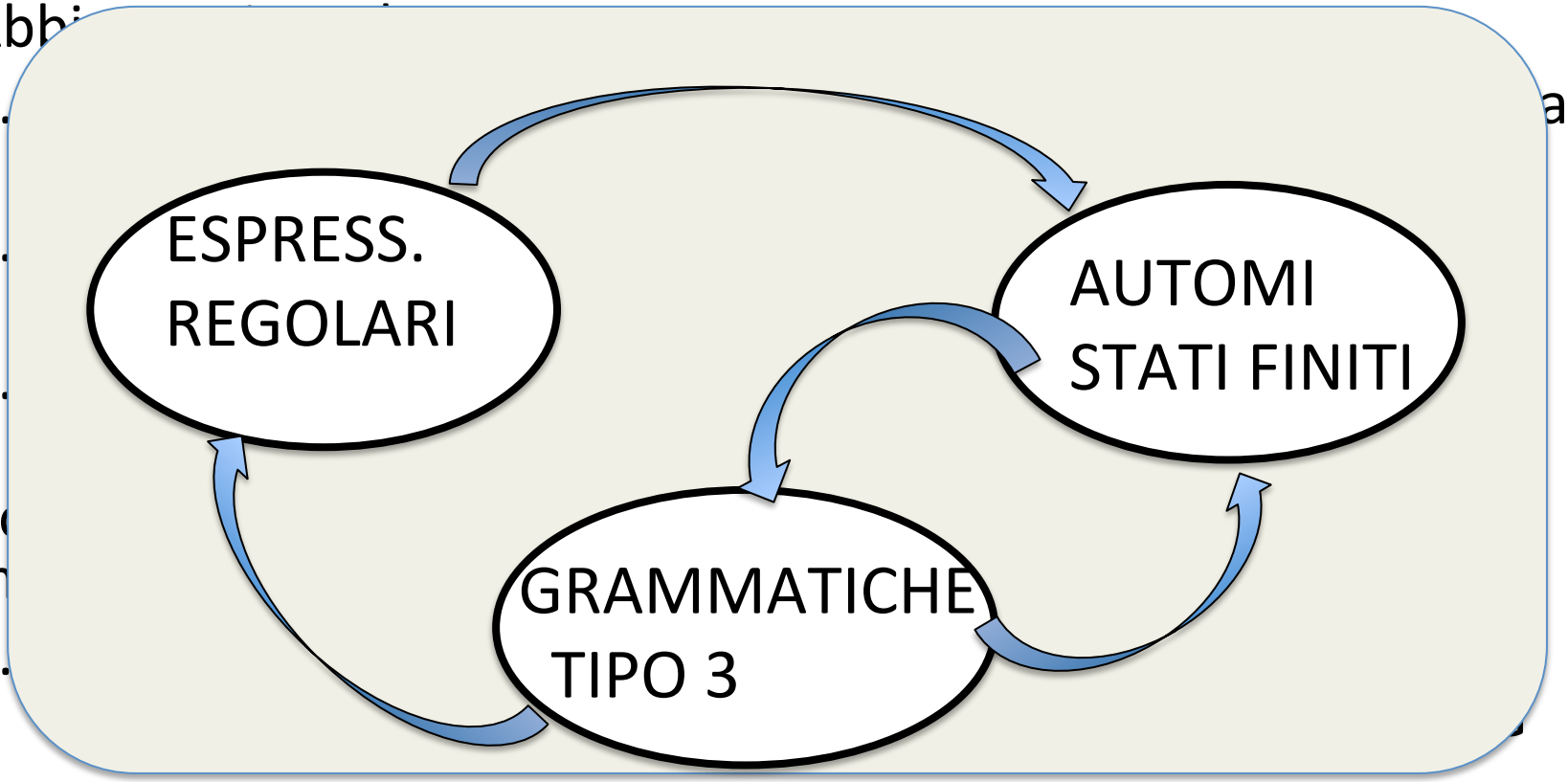
1.

2.

3.

P
m

4.



Conclusione: Automi stati finiti, grammatiche tipo 3 e espressioni regolari definiscono la stessa classe di linguaggi

Da grammatiche tipo 3 a espress. regolari

4. Per ogni grammatica regolare G esiste una espressione regolare che definisce lo stesso linguaggio generato da G

Sia data una grammatica G=<VT, VN, P, S> e supponiamo per semplicità che il linguaggio non contenga ε

Procediamo in due passi

1. Si trasforma la grammatica in un sistema di equazioni contenenti simboli terminali e variabili (le variabili assumono valore sull'insieme delle espressioni regolari).
2. Si risolvono le equazioni; Nota le equazioni non sono le usuali equazioni dell'algebra; e per questa ragione la soluzione è diversa

Da grammatiche tipo 3 a espress. regolari

1. Si trasforma la grammatica in un sistema di equazioni contenenti simboli terminali e variabili (le variabili assumono valore sull'insieme delle espressioni regolari)

- Ogni equazione è del tipo:

  <variabile>=<espressione regolare estesa>

In cui

- Le variabili assumono valore sull'insieme delle espressioni regolari
- una espressione regolare estesa è una espressione regolare in cui al posto di simboli terminali possiamo avere anche variabili.

  Ad esempio: A = (a+bA)* + bb

Da grammatiche tipo 3 a espress. regolari

1. **Sostituzione**

Si trasforma la grammatica in un sistema di equazioni contenenti simboli terminali e variabili (le variabili assumono valore sull'insieme delle espressioni regolari)

- Ogni equazione è del tipo:

  <variabile>=<espressione regolare estesa>

Esempio

Le produzioni A → aB | c danno A=aB+c

Le produzioni A→ aA | c danno A=aA + c

Intuizione: il simbolo '|' nelle grammatiche rappresenta l'alternativa ed è analogo all'operazone di "+" delle espressioni regolari

Da grammatiche tipo 3 a espress. regolari

- Ogni equazione è del tipo:

  <variabile>=<espressione regolare estesa>
- Per impostare il sistema procediamo nel seguente modo: raggruppiamo le produzioni che hanno lo stesso non terminale a sinistra
- ad ogni produzione (B1, B2,… Bn non terminali)

  A → a1 B1|a2 B2|....|an Bn|b1|b2|....|bm
- associamo l'equazione

  A = a1 B1+a2 B2+....+an Bn+b1+b2+....+bm

Da grammatiche tipo 3 a espress. regolari

- ad ogni produzione (B1, B2,… Bn non terminali)

  A → a1 B1|a2 B2|....|an Bn|b1|b2|....|bm

- associamo l'equazione

  A = a1 B1+a2 B2+....+an Bn+b1+b2+....+bm

Esempio. Alla grammatica S →aS|bB B →bB|c

corrisponde il sistema di equazioni

S = aS+bB B = bB+c

Da grammatiche tipo 3 a espress. regolari

2. Soluzione del sistema di equazioni

Le variabili corrispondono a espressioni regolari.

Quindi la soluzione del sistema richiede tecniche diverse da quelle usate nei normali sistemi di equazioni dell'aritmetica

Esempio. Alla grammatica $S \rightarrow aS|bB$ $B \rightarrow bB|c$

corrisponde il sistema di equazioni

$S = aS+bB$ $B = bB+c$

Nota che le definizione di S e B sono ricorsive

Per eliminare (risolvere) la ricorsione usiamo la **sostituzione**

Esempio di sostituzione $S \rightarrow aS$ ha come soluzione $S=a^*S$

Da grammatiche tipo 3 a espress. regolari

Esempio. Alla grammatica $S \rightarrow aS|bB$ $B \rightarrow bB|c$

corrisponde il sistema di equazioni ricorsivo

$S = aS+bB$ $B = bB+c$

Con soluzione $S= a^* + b(b^*+c)$

Intuizione del metodo di soluzione:

$B=bB +c$ definisce l'espressione regolare $B= b^* c$

Sostituendo otteniamo: $S= aS + b(b^*+c)$

Nota che $S= aS$ definisce $S= a^*S$

e quindi alla fine otteniamo $S= a^* + b(b^*+c)$

Da grammatiche tipo 3 a espress. regolari

Una volta fatti i passi 1 (sostituzione) e 2 (eliminazione della ricorsione) segue il passo 3 di semplificazione della formula

Con queste tre regole possiamo risolvere ogni sistema di equazioni lineari destre e determinare

- il linguaggio associato alla variabile corrispondente all'assioma.
- l'espressione regolare che descrive il linguaggio generato dalla grammatica data.

Da grammatiche tipo 3 a espress. regolari

Conclusione: i passi da fare sono:

1. (sostituzione) Scrivere un sistema derivato dalla grammatica di tipo 3:

- Ogni produzione un'equazione
- Ogni simbolo non terminale una variabile
- Ogni simbolo terminale una costante
- '|' diventa '+'

2. Eliminare la ricorsione (se necessario):

- $S = aS + X$ diventa $S = a^* X$
- $S = aS + b$ diventa $S = a^* b$ ($S=aS+ a$ diventa $S=a^*a = a^+$

3. Applicare il principio di sostituzione delle variabili e semplificare finché non si ottiene la sola equazione relativa all'assioma.

Alla fine: Il membro destro dell'equazione è l'espressione regolare che descrive il linguaggio generato dalla grammatica di tipo 3.

Da grammatiche tipo 3 a espress. regolari

Esempio. Sia data la grammatica con assioma A che genera il linguaggio delle stringhe che contengono un numero pari, anche 0, di a:

$A \rightarrow bA \mid aB \mid b$ $\qquad$ $B \rightarrow aA \mid bB \mid a$

Otteniamo il sistema

$A = bA + aB + b$ $\qquad$ $B = bB + aA + a$

- Si elimina la ricursione nella seconda equaz. $B = b^*(aA + a)$
- si sostituisce nella prima $A = bA + ab^*(aA + a) + b$
- si semplifica $A = bA + ab^*aA + ab^*a + b$
- si fattorizza $A = (ab^*a+b)A + ab^*a + b$
- e si termina eliminando di nuovo la ricursione

  $A = (ab^*a+b)^*(ab^*a + b) = (ab^*a+b)^+$

Esempio

Scrivere l'espressione regolare che descrive il linguaggio generato dalla grammatica di tipo 3

S → a S | b M

M → a M | b N | b

N → a N | a

Uno: Scrivere il sistema corrispondente

S = a S + b M

M = a M + b N + b

N = a N + a

Esempio

Uno: Scrivere il sistema corrispondente

$S = a\,S + b\,M$

$M = a\,M + b\,N + b$

$N = a\,N + a$

Due: Eliminare la ricorsione

$S = a^*\,b\,M$

$M = a^*\,(b\,N + b)$

$N = a+$

Esempio

Due: Eliminare la ricorsione

1. $S = a^* \, b \, M$
2. $M = a^* \, (b \, N + b)$
3. $N = a^+$

Tre: Sostituire le variabili e semplificare

- Sostituendo $N=a^+$ nella 2 otteniamo

  $M = a^* \, (ba^+ + b)$

- Semplificando otteniamo $M = a^*ba^*$
- Sostituendo $M = a^*ba^*$ nella 1 otteniamo

  $S = a^* \, ba^* \, ba^*$

Linguaggi non regolari

Come dimostrare che esistono linguaggi non regolari?

Una prova formale è presente nelle dispense

Nel seguito una intuizione

- Chiaramente tutti i linguaggi finiti (con un insieme finito di stringhe) sono regolari (esercizio)
- Se un linguaggio infinito L è deciso da un automa a stati finiti T, quest'ultimo deve necessariamente avere un numero finito n di stati.
- Un automa con un numero finito di stati non è in grado di contare

Formalizzando questa intuizione si può dimostrare che

Teorema Il linguaggio $L=\{a^n b^n, n>0\}$ non è regolare

(per riconoscere la stringa deve essere in grado di contare quante a ci sono e poi verificare se le b sono in ugual numero)