DFA, NFA, E\_NFA

PASSARE DA UN **NFA** AD UN **DFA**

(Pag 20 parte1.pdf)

Si fanno tutti i sottoinsiemi degli stati dell'NFA.

In tutto sono 2^n. (molti non sono raggiungibili dallo stato iniziale)

Lo stato iniziale del DFA equivalente è il sottoinsieme formato solo dallo stato che è lo stato iniziale nell'NFA.

Gli stati finali del DFA sono i sottoinsieme che contengono almeno uno stato finale nell'NFA.

La funzione di transizione del DFA è l'insieme di tutti gli stati raggiunti nell'NFA con un simbolo **a** dell'alfabeto, partendo da ogni singolo stato del sottoinsieme del DFA.

δ D (S, a) = **U** δ N (p, a)

p∈S

(guarda tabella pag 21, si ricava dal disegno pag 18)

RISPARMIARE LA CRESCITA ESPONENZIALE DEI SOTTOINSIEMI:

Si parte dallo stato iniziale, e guardando la funzione di transizione del DFA costruita col metodo sopra, si raggiungono tutti i sottoinsiemi raggiungibili applicando allo stato corrente raggiungibile uno per uno tutti i simboli dell'alfabeto

(vedi pag 21, parti da qo, con a = 0 raggiungi lo stato [qo,q1], con a = 1 rimani in q0, quindi [q0,q1] e q0 sono raggiungibili)

N.B: lo stato iniziale è sempre raggiungibile (base del teorema)

TEOREMA CRESCITA ESPONENZIALE DEGLI STATI: (pag 65 libro)

RISOLTO

(pag 26 slide)

Un NFA con n+1 stati deve avere almeno 2^n stati.

Nella figura viene riportato un NFA che ha un 1 come ennesimo simbolo dalla fine.

Questo NFA ha n+1 stati, gli ultimi n stati sono necessari per riconoscere i simboli dopo l’ennesimo 1 dalla fine.

Il DFA D deve poter accettare le stesse stringhe dell’NFA.

Il DFA deve ricordare gli ultimi n simboli che ha letto, per poter decidere se andare in uno stato accettante o no.

Per n bit, ci sono 2^n stringhe diverse.

A1a2a3a4………an

B1b2b3b4………bn

Quindi se il DFA ha meno di 2^n stati, allora deve avere almeno uno stato **q** in cui il DFA si trova dopo aver letto 2 sequenze differenti di n bit.

Dato che le 2 sequenze di bit sono diverse, devono differire per almeno un bit, e in quel bit che differiscono il DFA è nello stesso stato q.

CASO 1:

Se ho le stringhe:

1a2a3…an

0b2b3…bn

Allora lo stato q (quello che contiene tutti gli n bit, il DFA si deve ricordare gli ultimi n bit letti) deve essere sia accettante che no, dipende dal bit di partenza.

CASO 2:

Se i > 1, le 2 sequenze di n bit differiscono in una posizione nel mezzo, es:

a1a2a3…1…an

b1b2b3…0…bn

In questo caso, si aggiungono (i-1) zeri ad entrambe le stringhe, così si ricade nel caso 1, es:

a1a2a3…1…an000…0

b1b2b3…0…bn000…0

E lo stato q deve essere sia accettante che non, perché cambia l’ennesimo simbolo letto dalla fine.

EPSILON CHIUSURA - ECLOSE

(pag 30 guarda figura della chiusura)

Ogni stato appartiene alla propria ECLOSE

Passo di induzione: guarda def formale (facile)

DEFINIZIONE DELLA FUNZIONE DI TRANSIZIONE PER UN E\_NFA:

Passo base:

se la lettera è epsilon, la funzione di transizione è uguale alla ECLOSE di quello stato

Induzione:

p∈ δ ˆ (q,x) significa tutti gli stati p raggiunti da q con la stringa **x**

t∈δ(p,a) significa tutti gli stati t raggiunti da p con il carattere **a.**

La funzione di transizione di un E\_NFA di uno stato q è quindi l'insieme degli stati che fanno parte della ECLOSE degli stati t.

CONVERSIONE E\_NFA IN DFA

Gli stati del DFA sono i sottoinsiemi dell' E\_NFA, ogni sottoinsieme S contiene l'ECLOSE di ogni suo stato s.

Lo stato di partenza del DFA è il sottoinsieme formato dagli stati che fanno parte della ECLOSE dello stato iniziale dell' E\_NFA.

Gli stati finali del DFA sono tutti i sottoinsiemi che hanno almeno uno stato che è finale nell' E\_NFA

(vedi pag 32)

---

Per ogni sottoinsieme S, la sua funzione di transizione è il sottoinsieme formato dalla ECLOSE di tutti gli stati dell' E\_NFA raggiunti prendendo un singolo stato del sottoinsieme S del DFA e dandogli come input il carattere a.

---

(prova esempio pag 33)

Praticamente è come la conversione da NFA a DFA, solo che qui, per ogni stato trovato, si fa la ECLOSE di quello stato.

ESERCIZIO PAG 33, DA **E\_NFA** A **DFA**

Tu parti col disegno dell' E\_NFA da convertire in DFA.

Bisogna fare la funzione di transizione per il DFA con i sottoinsiemi.

Non si scrivono tutti i possibili sottoinsiemi, ma si usa il teorema di semplificazione degli stati.

La prima riga della funzione di transizione è la riga dello stato iniziale, poi le altre righe sono lo sviluppo della funzione di transizione sugli stati trovati nella prima riga, e così via finché non trovi tutti gli stati raggiungibili.

Per prima cosa, si trova lo stato iniziale del DFA, che è la ECLOSE dello stato iniziale dell' E\_NFA:

ECLOSE(qo) = [q0,q1]

Poi si da in pasto ad ogni singolo stato del sottoinsieme (in questo caso sia q0 che q1) ogni carattere dell'alfabeto, e si guarda ogni stato raggiunto nell' E\_NFA. Inoltre, per ogni stato raggiunto nell' E\_NFA, si fa la ECLOSE, e si scrivono tutti gli stati ottenuti in un nuovo sottoinsieme del DFA.

Alla fine, si disegna il DFA corrispondente, e viene uguale alla figura di pag 33.

**ESERCIZI**

Un NFA serve per non disegnare in uscita, su ogni stato, ogni carattere dell'alfabeto.

Se un carattere porta a più stati, si seguono tutte le strade contemporaneamente, e solo una strada alla fine porta allo stato giusto. (es 10 JFLAP)

Negli NFA, lo stato di partenza ha sempre un arco con tutti i caratteri dell'alfabeto verso se stesso.

Negli E\_NFA no, possono partire dallo stato iniziale tanti archi Epsilon.

MINIMIZZAZIONE DFA

1) si eliminano gli stati non raggiungibili

2) Si trovano gli stati equivalenti e distinguibili con l'apposita tabella,ogni **X** indica che la coppia di stati non è equivalente.

- Per prima cosa, si marcano tutte le coppie in cui p appartiene ad uno stato finale e q no

(tratteggia la diagonale dei quadretti per non ripetere le coppie)

- Per ogni cella non marcata, considera la coppia formata dai 2 stati.

Per ogni input dell'alfabeto, guarda la funzione di transizione.

Se i 2 stati trovati hanno la cella marcata, allora marca anche la cella vuota considerata prima.

-Continua così finché non marchi tutte le celle della tabella.

-Le celle che rimangono bianche indicano la coppia di stati equivalenti.

3) Ora raggruppi tutti gli stati equivalenti in partizioni separate (insieme K)

4) Lo stato iniziale è la partizione che contiene lo stato iniziale del DFA originale

5) Gli stati finali sono tutte le partizioni che contengono almeno uno stato finale del DFA originale

6) La funzione di transizione cambia: per ogni input dell'alfabeto, tutti gli stati di una partizione devono puntare ad uno stato di un'altra partizione, non a stati di partizioni separate.

(fai l'esempio in 4.1-minfa.pdf)

ESPRESSIONI REGOLARI

**Chiusura di Kleene:**

E' l'insieme di tutte le stringhe di un alfabeto.

O dato un linguaggio, è l'insieme di tutte le stringhe che si possono formare con quei caratteri.

Se al posto della chiusura di Kleene hai una potenza (es. 3), si considerano tutte le stringhe prendendo solo 3 stringhe del linguaggio (ogni stringa può essere sia un carattere che più caratteri).

Ad esempio, se un linguaggio contiene solo caratteri, si prendono tutte le stringhe di lunghezza 3.

Esempi: ([] è l’insieme vuoto)

[]^0 = epsilon

[]^i = []

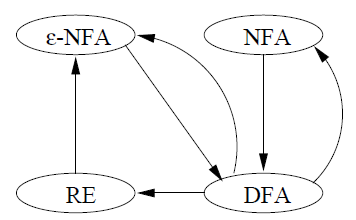
[]^\* = epsilon

(01)\* +(10)\* +0(10)\* +1(01)\*

Sono tutte le stringhe che iniziano per 0 e finiscono per 1, iniziano per 1 e finiscono per 0, iniziano per 1 e finiscono per 1 e iniziano per 0 e finiscono per 0.

(vedi pag 4 di lezione 02.pdf)

EQUIVALENZA DI FA ED ESPRESSIONI REGOLARI



Un linguaggio regolare può essere espresso in questi 4 modi.

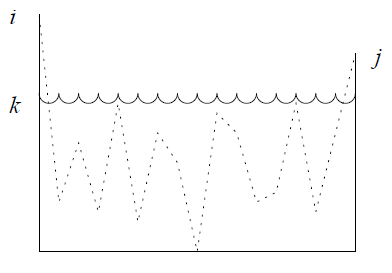
Per dimostrare che lo stesso linguaggio regolare si può esprimere anche con un’espressione regolare (RE), bisogna trovare un modo per costruire un’espressione regolare da un DFA e costruire un E\_NFA da una RE.

TEOREMA 3.4:

Per ogni DFA A = (Q;\_; \_; q0; F) esiste una espressione regolare R, tale che L(R) = L(A).

Sia R(k)ij una espressione regolare che descrive l'insieme delle

etichette di tutti i cammini in A che vanno dallo stato i allo stato j passando solo attraverso gli stati [1……k]



Per costruire Rij(k) si parte da k = 0 (base del teorema), poi per induzione si arriva fino a k = n, quindi non c’è più nessuna restrizione per la RE, può rappresentare i percorsi formati da tutti gli stati in quanto gli stati non sono più grandi di n.

**Base:**

K = 0.

Quindi se k = 0 non ci sono stati intermedi tra gli stati i-j.

Ci sono 2 casi:

1. C’è un arco che collega direttamente i a j
2. Lo stato i e j sono o stesso stato, il percorso ha lunghezza 0

Bisogna vedere adesso nel DFA tutti i simboli di input che portano dallo stato i allo stato j.

1. Se non c’è nessun simbolo a, Rij(0) = []
2. Se c’è un solo simbolo a, Rij(0) = **a**
3. Se ci sono tanti simboli, Rij(0) = **a1 + a2 + … + ak**

Se i = j, lo stato è lo stesso, e bisogna aggiungere epsilon.

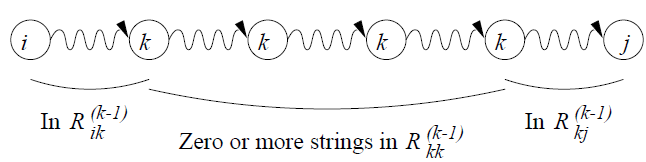
1. Se non c’è nessun arco, Rij(0) = epsilon
2. Se ci sono uno o più archi, Rij(0) = epsilon + a1 + a2 + … + ak.

**Induzione:**

Nell’induzione k > 0, gli stati intermedi non sono maggiori di k.

Ci sono 2 possibili casi da considerare:

1. Se il percorso non passa per tutti gli stati k, le etichette del percorso sono rappresentate dal linguaggio Rij(k-1) (può essere ciclico)
2. Se il percorso passa almeno per una volta per uno stato k, si divide il percorso in 3 pezzi:  
     
   Il primo pezzo contiene tutti gli stati prima di arrivare allo stato k,  
     
   Il secondo contiene tutti i percorsi che vanno dallo stato k a se stesso, una o più volte,  
     
   Il terzo va da k allo stato finale j.  
     
   La RE quindi è:  
     
   Rik(k-1) (Rkk(k-1))\* Rkj(k-1)

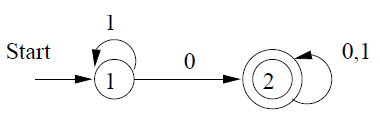


Bisogna combinare questi 2 casi facendo l’unione delle 2 RE:

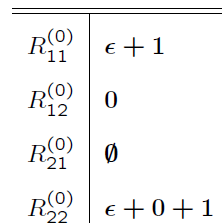
Rij(k) = Rij(k-1) + Rik(k-1) (Rkk(k-1))\* Rkj(k-1)

ESEMPIO: CONVERTIRE IN RE IL SEGUENTE DFA:

Questo automa riconosce tutte le stringhe che hanno almeno uno 0:



Si parte con il caso base:



(gli stati i-j si considerano per ogni coppia di stati del DFA)

Con k = 0, nel caso base, significa che non ci sono gli stati intermedi, però si può andare da uno stato a un altro differente

Si continua poi con il passo di induzione:

Devo calcolare adesso Rij(1):

Rij(1) = Rij(0) + Ri1(0) (R11(0))\* R1j(0)

(nel mezzo ci va R11 perché k = 1)

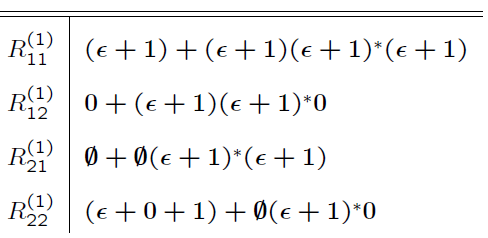
Rij(1) deve essere calcolato per ogni stato i-j del DFA:

R11(1) = R11(0) + R11(0) (R11(0))\* R11(0)

R12(1) = R12(0) + R11(0) (R11(0))\* R12(0)

R21(1) = R21(0) + R21(0) (R11(0))\* R11(0)

R22(1) = R22(0) + R21(0) (R11(0))\* R12(0)



Semplificazioni:

(e + 1)\* = 1\*

(e + 1)1\* = 1\*

1.1\* = 1\*

Le espressioni di prima si possono quindi semplificare:

R11(1) = 1\*

R12(1) = 0 + 1\*0 = 1\*0

R21(1) = []

R22(1) = e + 0 + 1

L’ultimo passo di induzione è Rij(2) (k = 2 = n)

R11(2) = R11(1) + R12(1) (R22(1))\* R21(1)

R12(2) = R12(1) + R12(1) (R22(1))\* R22(1)

R21(2) = R21(1) + R22(1) (R22(1))\* R21(1)

R22(2) = R22(1) + R22(1) (R22(1))\* R22(1)

R11(2) = 1\* + 1\*0 (e + 0 + 1)\* [] = 1\*

R12(2) = 1\*0 + 1\*0 (e + 0 + 1)\* (e+0+1) = 1\*0(0 + 1)\*

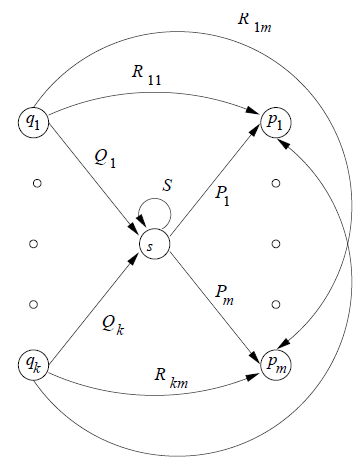
R21(2) = [] + …[] = []

R22(2) = e+0+1 + (e+0+1) (e+0+1)\* (e+0+1) = (0 + 1)\*

La RE del DFA è quindi la R12(2) = 1\*0(0 + 1)\*

TECNICA DI ELIMINAZIONE DEGLI STATI:

Per evitare di scrivere tutte le espressioni regolari Rij(k), si trasforma l’automa etichettando gli archi con le espressioni regolari:



In questo automa, gli stati q sono i predecessori di s, gli stati p sono i successori di s. Lo stato generico s sta per essere eliminato.

Se si vuole eliminare lo stato s, i predecessori q sono collegati direttamente ai successori p, tramite l’espressione Qi S\* Pj.

Esempio:

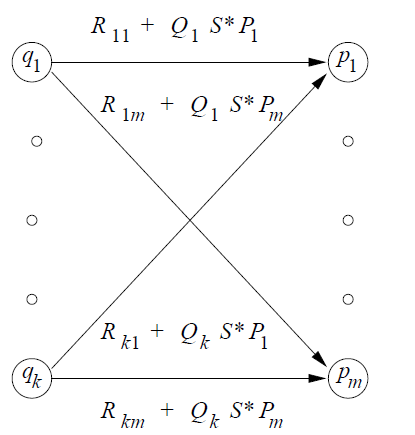
Rq1pm = Q1 S\* Pm

Rqkp1 = Qk S\* P1

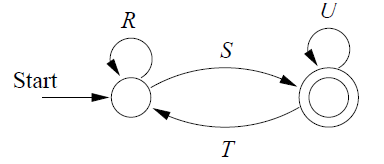
Si cancella poi s.

Se quando si cancella s ci sono più di un arco, si fa un unico arco con l’unione delle RE.

Questa riduzione si fa per ogni stato q finale. Si possono eliminare tutti gli stati tranne lo stato q correntemente in uso e lo stato iniziale q0.

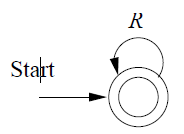


Seguendo queste riduzioni per ogni stato finale, siamo rimasti o con la forma:





O con la forma:





L’espressione regolare finale è data dalla somma (unione) di tutte le espressioni derivate dall’automa ridotto per ogni stato finale:



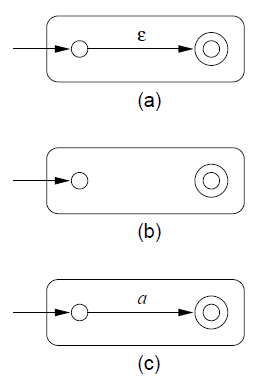
Guarda Es pag 47 della parte1.pdf

DA ESPRESSIONI REGOLARI A E\_NFA

Teorema 3.7:

**Base:**

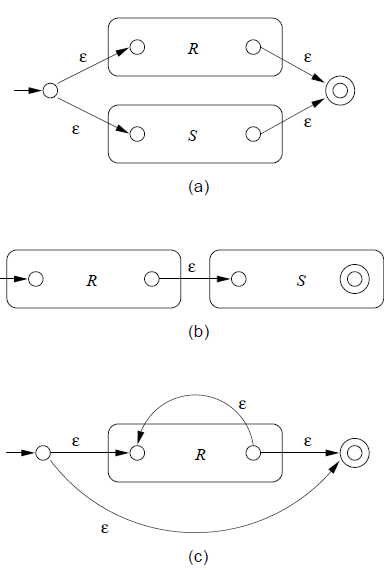
Automi per epsilon, insieme vuoto e **a**



**Induzione:**

Automi per:

R + S, R.S, R\*



Guarda es pag 52

PROPRIETA’ DEI LINGUAGGI REGOLARI:

Pumping Lemma,

Chiusura,

Decisione,

Tecniche di minimizzazione.

PUMPING LEMMA PER LINGUAGGI REGOLARI

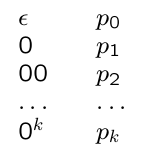
DESCRIZIONE INFORMALE:

Linguaggio L01 = “stesso numero di 0 seguiti da stesso numero di 1”

NON è regolare.

Se fosse regolare, esisterebbe un automa che ha k stati che riconosce tutte le sue stringhe.

Supponiamo che il DFA legga k zeri, considerando anche epsilon, quindi legge k+1 input.



Dato che il DFA ha solo k stati, esiste uno stato q in cui il DFA è nello stesso stato per 2 indici diversi di 2 zeri,

ad esempio 0i e 0j.

Supponiamo ora che, quando il DFA è nello stato q, inizi a ricevere i 1.

Nello stato q, il DFA NON si può ricordare se ha ricevuto prima j zeri o i zeri, quindi possiamo ingannare l'automa.

Se aveva ricevuto i zeri, lo possiamo fare sbagliare NON facendo accettare il linguaggio, se aveva ricevuto j zeri lo possiamo fare sbagliare facendo accettare il linguaggio.

DIMOSTRAZIONE FORMALE

**Teorema 4.1:**

Supponiamo che L sia un linguaggio regolare.

n è una costante, tale che per ogni stringa w in L tale che |w| > n, la stringa w può essere divisa in 3 stringhe xyz:

w = xyz, tali che:

1) y != epsilon

2) |xy| <= n

3) per ogni k >= 0, la stringa x y^k z appartiene al linguaggio

(in questo punto, se k = 0 y diventa epsilon e si cancella, la stringa xz deve appartenere al linguaggio)

**Prova:**

Supponiamo che L sia regolare. Allora L è il linguaggio di un certo DFA “A”, L = L(A).

Supponiamo che A abbia n stati.

Consideriamo tutte le stringhe del linguaggio tali che |w| > n, es w = a1a2a3...am, m >= n

Per i che va da 0 a n definiamo lo stato pi come



dove q0 è lo stato iniziale del DFA.

Lo stato pi indica in quale stato l'automa si trova dopo aver letto i input. Quindi p0 = q0.

Dato che m > n, il DFA deve leggere almeno n+1 input, quindi non è possibile che tutti gli stati pi siano differenti, perché il DFA ha solo n stati, è possibile trovare quindi 2 interi i e j tali che:

0 <= i < j <= n, tali che pi = pj (sono lo stesso stato)

Ora possiamo scomporre w = xyz in questo modo:

1) x = a1a2....ai

2) y = ai+1 ai+2 … aj

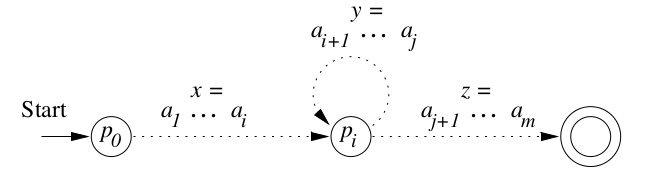
3) z = aj+1 aj+2 … am

(x può essere vuota se i = 0,

y per definizione no,

z può essere vuota se j = n = m)

La relazione fra stringhe e stati si vede nella seguente figura:



Quindi x y^k z appartiene al linguaggio, per ogni k >= 0, perché:

se k = 0, la stringa xz viene accettata, perché il DFA va da p0 a pi, e dato che pi = pj, va da pj nello stato accettante.

Se k > 0, A va da p0 a pi, cicla per k volte nello stato pi, e poi va nello stato accettante.

(può essere scelto un qualsiasi k > 0, non c'è un limite superiore)

Quindi il linguaggio contiene la stringa x y^k z.

(Se si trova un controesempio, allora il linguaggio non è regolare).

ESERCIZI:

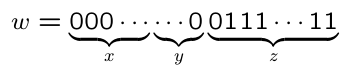
USANDO IL PUMPING LEMMA, DIMOSTRA CHE IL LINGUAGGIO NON È REGOLARE

Linguaggio con n zeri seguito da n uno:

Supponiamo che n sia costante.

Possiamo dividere la stringa w = 0^n 1^n in qualsiasi modo, dal pumping lemma sappiamo solo che y != epsilon e |xy| < n.

Questa informazione è fondamentale, in quanto sappiamo dal pumping lemma che la stringa xz deve appartenere al linguaggio regolare.



Ma xz non può appartenere al linguaggio, perché il numero di zeri di x è < del numero di 1 di z, perché y non può essere vuota.

-----------

Linguaggio formato da stringhe di 1, la cui lunghezza p è un numero primo

Scegliamo un numero primo p >= n + 2

Dividiamo w in questo modo, tali che y != epsilon e |xy| <= n

|y| = m, |xz| = p – m

Il pumping lemma dice che per ogi k >= 0, la stringa x y^k z appartiene al linguaggio.

Consideriamo la stringa x y^(p-m) z, la sua cardinalità deve avere lunghezza = un numero primo, per definizione del linguaggio:

|x y^(p-m) z| = |xz| + (p-m)|y| = p–m + (p–m)\*m = (p-m)\*(1+m)

Sembra che non sia un numero primo perché è il prodotto di 2 fattori, però se un fattore è 1 l'altro potrebbe essere primo.

Dato che y != epsilon, allora |y| >= 1, quindi m >=1, quindi (m+1) è > di 1.

Per scelta del numero primo, p-m è > 2, perché:

abbiamo scelto un numero primo p >= n + 2

|y| = m <= |xy| <= n, quindi m <= n, quindi p – m è >= 2, e non è 1.

Quindi il prodotto fra i 2 fattori non può essere primo, perché entrambi sono diversi da 1.

PROPRIETÀ DI CHIUSURA DEI LINGUAGGI REGOLARI:

Se applico una di queste operazioni ad un linguaggio regolare, ottengo un altro linguaggio regolare:

• Unione: L ∪ M

• Intersezione: L ∩ M

• Complemento: L (complemento, sbarra sopra)

• Differenza: L \ M

• Inversione: L Reverse = {w Reverse : w ∈ L}

• Chiusura: L∗ .

• Concatenazione: L.M

**Teorema 4.4**.

Per ogni coppia di linguaggi regolari L e M , L ∪ M

e’ regolare.

**Prova**.

Sia L = L(E) e M = L(F ). Allora L(E + F ) = L ∪ M per

definizione.

**Teorema 4.5.**

Se L e’ un linguaggio regolare su Σ, allora anche

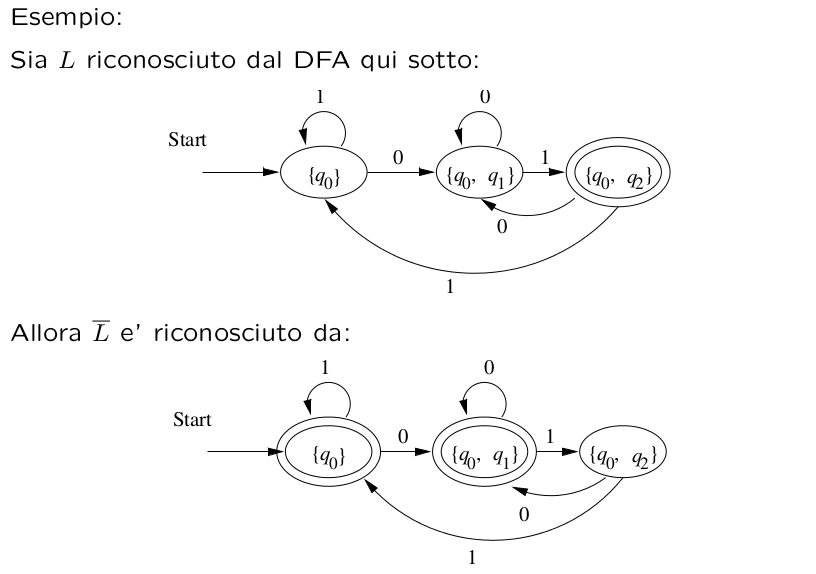
L = Σ ∗ \ L e’ regolare.

**Prova.**

Sia L riconosciuto da un DFA

A = (Q, Σ, δ, q 0 , F ).

Sia B = (Q, Σ, δ, q 0 , Q \ F ). Allora L(B) = L, perché è il linguaggio di un DFA.



**Teorema 4.8.**

Se L e M sono regolari, allora anche L ∩ M e’

regolare.

**Prova**.

Per la legge di DeMorgan, L ∩ M = (Lc ∪ Mc)c

Sappiamo gia’ che i linguaggi regolari sono chiusi rispetto al complemento e all’unione.

**Teorema 4.10**.

Se L e M sono linguaggi regolari, allora anche

L \ M e’ regolare.

**Prova**.

Osserviamo che L \ M = L ∩ Mc .

Sappiamo gia’ che i linguaggi regolari sono chiusi rispetto al complemento e all’intersezione.

**Teorema 4.11.**

Se L e’ un linguaggio regolare, allora anche L R

e’ regolare.

**Prova 1**:

Sia L riconosciuto da un FA A.

Modifichiamo A per renderlo un FA per L Reverse (L^R) :

1. Giriamo tutti gli archi.

2. Rendiamo il vecchio stato iniziale l’unico stato finale.

3. Creiamo un nuovo stato iniziale p 0 , con δ(p 0 , ) = F

(i vecchi stati finali).

Praticamente si crea uno stato iniziale con transazioni epsilon sui vecchi stati finali, che diventano tanti stai iniziali.

Alla fine si ottiene un E\_NFA che accetta tutte le stringhe inverse del linguaggio di partenza.

PROPRIETÀ DI DECISIONE DEI LINGUAGGI REGOLARI:

salto tutti i costi con gli ordini di grandezza, test linguaggio vuoto e l'appartenenza, da fare

(pag 76 parte1.pdf)

salto anche teoria della minimizzazione dei DFA