LINGUAGGI REGOLARI

CREA LINGUAGGIO REGOLARE DA UN DFA

ESERCITAZIONE 10\_3.TXT:

Alla fine della conversione, trovi sempre uno dei 2 stati spiegati nella pag 46 della parte1.pdf, alcuni archi possono essere l’insieme vuoto, ciè non esistere.

Per non sbagliare, applica sempre la prima formula se lo stato finale è diverso dallo stato iniziale, la seconda se coincidono.

Usa la tecnica di eliminazione degli stati (pag 44 parte1.pdf) per eliminare uno stato intermedio con un arco che gli fa da ciclo.

Esercizio 1 della cartella Esempi, converti da NFA a RE:

Quando ci sono 2 archi che vanno da uno stato ad un altro, si fa l’unione delle 2 espressioni regolari.

Se c’è un arco che ritorna indietro e si cancella lo stato nel mezzo, si scrive la RE concaenata con quella che ritorna nell’ultimo stato (in questo caso aa).

Esercizio 2:

L’arco che torna dallo stato accettante allo stato iniziale non si cancella come nell’esercizio di prima, a si usa nella formula di pag 46.

Esercizio 3:

Usa la tecnica di eliminazione degli stati (pag 44 parte1.pdf) per eliminare uno stato intermedio con un arco che gli fa da ciclo.

4. Definire una grammatica libera dal contesto (non lineare destra) sull'alfabeto {a,b,c} per i seguenti linguaggi:

**a- linguaggio delle stringhe (abc)^n (cba)^m (con n=m)**

è la stessa grammatica dei palindromi.

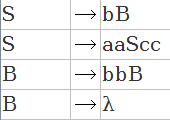
È un albero che cresce centralmente su S:

****

**b- linguaggio delle stringhe a^n b^m c^n con n pari e m dispari**

Prima genero il caso base con solo le b dispari, poi se ci sono anche le a e le c, le metto prima con la ricorsione su S.

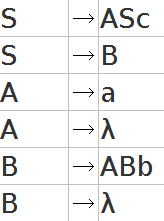
è un albero che cresce centralmente su b



**c- linguaggio delle stringhe a^n b^m c^p in cui n < m+p**

è un albero che cresce verso sinistra:

le c rimangono nella parte più a destra e alta dell'albero, poi vengono le b e poi le a.



**c2- come prima, però i != j + k:**



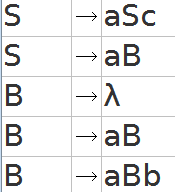
ci sono quindi 2 casi:

1) i < j + k

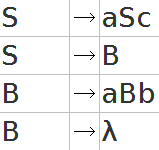
2) i > j + k

Si fa l'unione di 2 grammatiche, di cui quella al punto 1 è la grammatica dell'esercizio precedente.

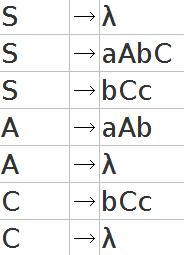
Caso 2: i > j + k



**c3- caso in cui i = j + k**

****

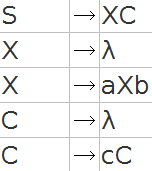
**c4- caso in cui: j = i + k**



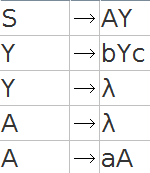
**c5- caso in cui: i = j OR j = k**

Si fa l'unione dei 2 casi:

caso 1: i = j



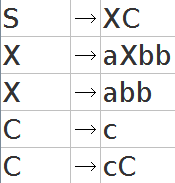
caso 2: j = k



La grammatica finale ha una nuova variabile che genera o S1 o S2.

**C6- trova la grammatica del linguaggio:**

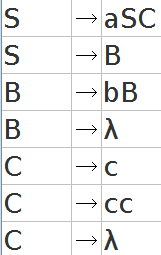
L = {a^n b^2n c^k | n, k >= 1}



**c7- Trova la grammatica del linguaggio**

L = {a^k b^m c^n | 2k >= n}

è il linguaggio in cui il numero delle c è al massimo il doppio del numero delle a, per ogni a quindi posso inserire al massimo 2 c (ma anche 0 o una sola):

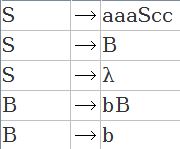


**c8- Trova la grammatica per questo linguaggio:**

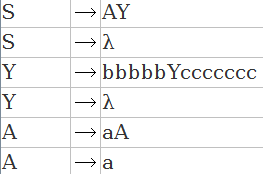
{a^n b^m c^k | 2n = 3k or 5k = 7m}

Si distinguono i 2 casi, in cui si fa poi l'unione.

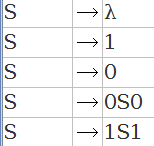
Caso 1: 2n = 3k



caso 2: 5k = 7m



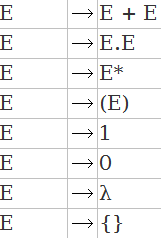
**d- la grammatica dei palindromi in {0,1}:**



**e- la grammatica delle espressioni regolari in {0,1}:**

Nelle espressioni regolari ha senso anche concatenare l'insieme vuoto e un'espressione (il risultato è l'insieme vuoto).

Anche con epsilon.

****

**f- la grammatica di un semplice linguaggio di programmazione, dove le variabili iniziano con {a,b} e possono essere seguite da {a,b,0,1}. Le operazioni disponibili sono il + e il \*:**

Per evitare l'**ambiguità** delle operazioni, definisco 3 tipi diversi di variabili:

1) una variabile non può essere spezzato ne da un + ne da un \*.

Es:

i nomi delle variabili, un'espressione fra parentesi

2) un fattore è un'espressione che non può essere spezzata da un +, può essere spezzata solo se viene preceduta da un \* (con associatività a sinistra)

Es:

a + (a \* a)

(a \* a) \* a

a \* ba

a \* (E)

a \* a + b

3) un termine può essere spezzato da un \* (sia prima che dopo), e da un + precedente

Es:

a + a

a + E

a + (E \* b)

(a + b) + c

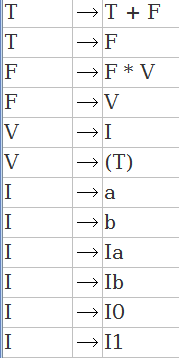
Indico le variabili con V, i fattori con F e i termini con T.

Praticamente, si parte dalle espressioni più generiche, in questo caso i termini, che possono essere sommati fra di loro, dove un termine è o un fattore, o una somma fra termine e fattore (la ricorsione su termine permette di avere più somme di fila, ad esempio).

Un fattore può essere o una variabile, o un prodotto fra fattore e variabile (anche qui la ricorsione permette di avere altri \* in precedenza)

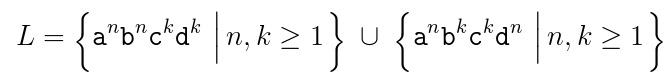
Una variabile è appunto un identificativo o un termine tra parentesi.

(Nelle operazioni, la ricorsione va sempre a sinistra, prima del segno, perché l'associatività è a sinistra.)



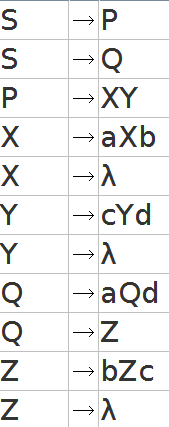
**f2- grammatica inerentemente ambigua:**

è conosciuta in letteratura:



è inerentemente ambigua perché per ogni grammatica si possono trovare almeno 2 parse tree.

Una grammatica possibile fa l'unione dei 2 linguaggi:



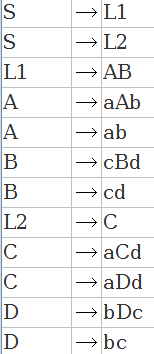
**g- la grammatica che fa l'unione dei 2 linguaggi:**

****

Per fare l'unione, si fanno le 2 grammatiche separatamente, poi si aggiunge un nuovo simbolo iniziale che produce o la prima grammatica o la seconda.

Per la prima grammatica, produco le ab nello stesso numero concatenate con le cd nello stesso numero.

Per la seconda grammatica, produco prima le ad nello stesso numero, poi le bc. Per la ricorsione di ad si usa sempre la stessa variabile, non se ne crea una nuova.



**h- Trova una grammatica per questo linguaggio:**



TIP:

La condizione può essere scritta come:

NOT (i = j AND j = k)

Quindi, per la legge di De Morgan:

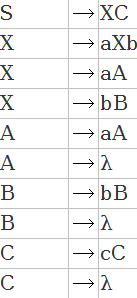
NOT (i = j AND j = k) =

NOT(i = j) OR NOT (j = k) =

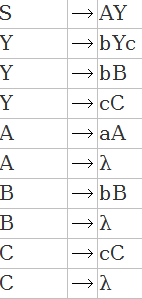
(i != j) OR (j != k)

La grammatica è l'unione di 2 linguaggi:

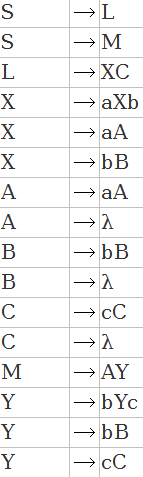
1) i != j:



2) j != k

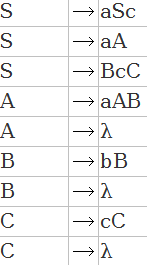


La grammatica finale quindi è:

****

**h2- Trova una grammatica per questo linguaggio:**

{a^i b^j c^k | i != k }



è un po' diverso dalle 2 grammatiche sopra, perché le 2 grammatiche sopra accettano entrambe la stringa ac (stesso numero di a e di c).

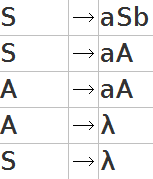
La B fa una ricorsione delle b, da 0 a quante ne vuoi.

S genera sempre una coppia bilanciata di ac, per terminare però deve generare altre a o altre c, quante volte vuole.

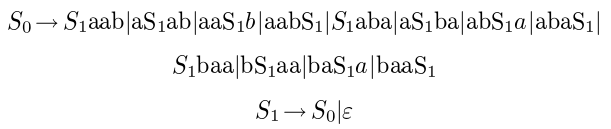
In altre parole, o fa vincere il numero delle a o fa vincere il numero delle c.

**i- Trova la grammatica per questo linguaggio:**





**j- trova la grammatica in cui il numero delle a è il doppio del numero delle b (MIXED, in qualsiasi ordine)**



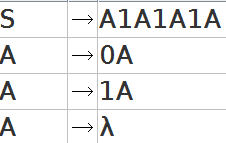
oppure, meglio:



si fanno tutte le combinazioni di 110, 101, 011, e dopo ogni terminale si mette una ricorsione S.

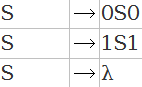
**k- Trova la grammatica per:**





**l- Trova la grammatica per:**

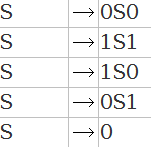




Se doveva essere dispari, S doveva generare anche 0 e 1 separatamente, e non avere la produzione epsilon.

**m- trova la grammatica per:**



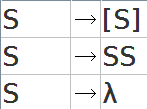


**-n trova la grammatica per il linguaggio vuoto:**



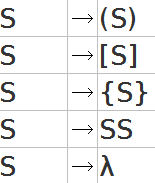
**-o grammatica per il linguaggio delle parentesi quadre bilanciate, es:**

[ ] [ [ [ ] [ ] ] [ ] ]

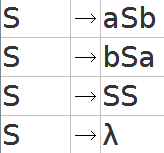


**-p grammatica per parentesi tonde, quadre e graffe bilanciate, es:**

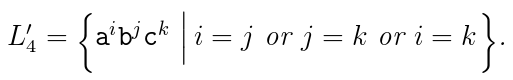
{([])}[]({})



**q- trova la grammatica per il linguaggio in {a,b}\* dove il numero delle a = numero delle b**

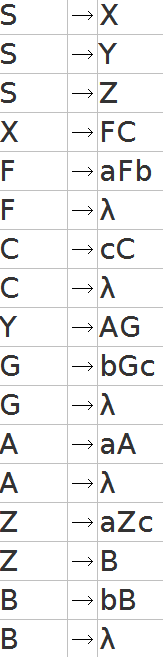


**r- trova la grammatica per:**

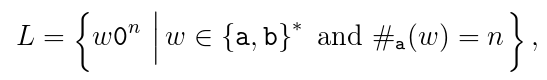


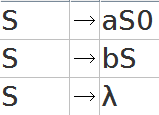
Si fa con l'unione di 3 linguaggi:

X: i = j, Y: j = k, Z: i = k

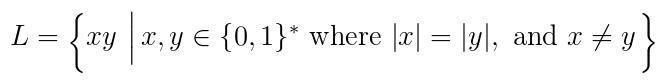


**s- trova la grammatica per:**

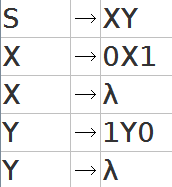




**t- trova la grammatica per:**



è l'unione di 2 casi: prima gli 0 o prima gli 1



CHIUSURA DEI CFL

**Prova della chiusura rispetto all'unione:**

Date 2 grammatiche:

G1(V1,T,P1,S1) G2(V2,T,P2,S2)

Allora si definisce una nuova grammatica G3(V3,T,P3,S3)

dove:

V3 = V1 U V2 U S3

P3 = P1 U P2 U {produzioni di S3}

S3 => S1 | S2

Allora possono essere generate sia le stringe di G1 che di G2, perché:

S3 S1 w, per ogni stringa di L(G1)



S3 S2 w, per ogni stringa di L(G2)

**Prova della chiusura della concatenazione:**

Date 2 grammatiche:

G1(V1,T,P1,S1) G2(V2,T,P2,S2)

Allora si definisce una nuova grammatica G3(V3,T,P3,S3)

dove:

V3 = V1 U V2 U S3

P3 = P1 U P2 U {produzioni di S3}

S3 => S1 S2

Ogni stringa w = w1w2, dove w1 appartiene a L(G1) e w2 appartiene a L(G2), si può ottenere in questo modo:

S3 S1 S2

S1 w1



S2 w2

**Prova della chiusura della stella di Kleene:**

Date 2 grammatiche:

G1(V1,T,P1,S1)

Allora si definisce una nuova grammatica G3(V3,T,P3,S3)

dove:

V3 = V1 U S3

P3 = P1 U {produzioni di S3}

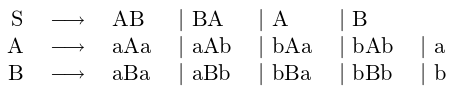
S3 => S1 S3 | epsilon

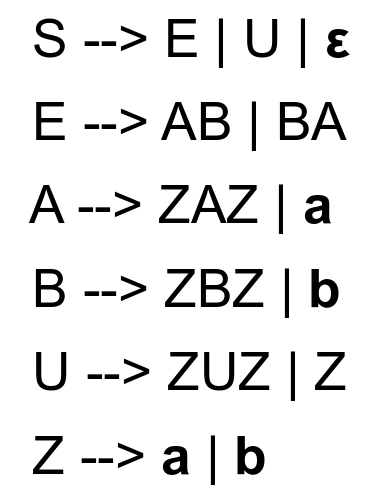
Ogni stringa w\* che appartiene a L(G1)\* può essere prodotta con n concatenazioni della stringa w:

w\* = w1 w2 w3 w4 … wn,

dove n può essere anche zero (in questo caso si produce epsilon)

**not form ww:**





sicuramente le stringhe dispari non sono ww

quelle pari devono essere diverse

**operazioni delle espressioni regolari in {0,1} con**

**T = {0,1,(,),+,\*,phi,epsilon}**

S -> S+S | SS | S\* | (S) | 0 | 1 | phi | e