CAPITOLO 1

Dato l’alfabeto binario X = {a,b}, elencare in ordine crescente di lunghezza tutte le parole su X di lunghezza minore o uguale a 3.

X\*= {lambda, a, b, aa, bb, ab, ba, aaa, aab, abb, bbb, bba, baa, aba, bab}

Data la parola w = ababba, determinare:

♦ la lunghezza di w (|w|)

♦ l’insieme dei prefissi di w (Prefix(w))

♦ l’insieme dei suffissi di w (Suffix(w))

♦ l’insieme delle sottostringhe di w (Substring(w))

♦ le seguenti potenze di w: 𝑤0 , 𝑤2 , 𝑤3 con le relative lunghezze

|w| = 6

Prefix(w) = {lambda, a, ab, aba, abab, ababb, w)

Suffix(w) = {lambda, a, ba, bba, abba, babba, w}

Substring(w) = {lambda, a, b, ab, ba, bb, aba, bab, abb, bba, abab, babb, abba, w}

w^0 = lambda |w^0| = 0

w^2 = ababbaababba |w^2| = 12

w^3 = ababbaababbaababba |w^3| = 18

Dato l’alfabeto ternario X = {a,b,c}, elencare in ordine crescente di lunghezza tutte le parole su X di lunghezza minore o uguale a 2.

X\* = {lambda, a, b, c, ab, aa, ac, ab, bb, ba, bc, cc, ca, cb}

Data la parola w = abcba, determinare

♦ la lunghezza di w (|w|)

♦ l’insieme dei prefissi di w (Prefix(w))

♦ l’insieme dei suffissi w (Suffix(w))

♦ l’insieme delle sottostringhe di w (Substring(w))

♦ le seguenti potenze di w: 𝑤0 , 𝑤2 , 𝑤3con le relative lunghezze

|w| = 5

Prefix(w) = {lambda, a, ab, abc, abcb, w)

Suffix(w) = {lambda, a, ba, cba, bcba, w}

Substring(w) = {lambda, a, b, c, ab, bc, cb, ba, abc, bcb, cba, abcb, bcba, w}

w^0 = lambda |w^0| = 0

w^2 = abcbaabcba |w^2| = 10

w^3 = abcbaabcbaabcba |w^3| = 15

Dato l’alfabeto binario X = {a,b}, determinare le seguenti potenze di X: 𝑋 0 , 𝑋 1 , 𝑋 2 , 𝑋 3 .

X^0 = {lambda}

X^2 = {ab, ba, aa, bb}

X^3 = {aba, abb, baa, bab, aaa, aab, bba, bbb}

Dato l’alfabeto ternario X = {a,b,c}, determinare le seguenti potenze di X: 𝑋 0 , 𝑋 1 , 𝑋 2 , 𝑋 3

X^0 = {lambda}

X^2 = {aa, ab, ac, ba, bb, bc, ca, cb, cc}

X^2 = {aaa, aab, aac, aba, abb, abc, aca, acb, acc, baa, bab, bac, bba, bbb, bbc, bca, bcb, bcc, caa, cab, cac, cba, cbb, cbc, cca, ccb, ccc}

CAPITOLO 2

Determinare una grammatica generativa o a forma di frase G tale che L(G)=L

L={ w € {a,b}\* | w = a^n b^n} = {ab,aabb,aaabbb, …}

G={X,V,S,P}

X = {ab}

V = {S}

Passo base: ab€L ----> Passo induttivo: se A€L allora aAb€L

P = {S --> ab, S --> aSb}

Sia data la seguente grammatica: S—> A|B, A --> aA|a, B --> bB | b

Determinare L(G).

G = {X, V, S, P}

X = {a, b}

V = {S,A,B}

P = {S—> A|B, A --> aA|a, B --> bB | b}

Si distinguono G’ e G’’ ove: G’ = ({a}, {A}, A, {A-->aA | a}), L(G’) = {w€{a}\* | w = a^n, n>0}, G’’ = ({b}, {B}, B, {B-->bB | b}), L(G’’) = {w€{b}\* | w = b^n, n>0}

L(G) = L(G’) U L(G’’)

Esercizio 2.3 Pag. 48 Fatto dal prof

1. Determinare una grammatica che genera il seguente linguaggio: L = {w € {a,b}\* | a^n b^2n | n>0}. E dimostrare questo risultato
2. Di che tipo è la grammatica che genera L?

Svolgimento 1:

Vogliamo trovare una grammatica G tale che L(G)=L.

L = {a^n b^2n | n>0} = {abb,aabbbb,aaabbbbbb…,}

Passo base: abb € L

Passo induttivo: se A€L allora aAbb € L

Il passo base si traduce in S --> abb

Il passo induttivo si traduce in S --> aSbb

Esercizio 2.2 Pag.48 al contrario fatto dal prof

L = {w € {0,1}\* | #(0,w)=#(1,w)} ove #(x,w)=numero di volte che x compare in w

L = {01,10,0011,1100,0110,1001,0101,1010,…}

Definizione per induzione:

* Passo base: 01€L, 10€L
* Passo induttivo: Se A€L allora 0A1€L e 1A0€L
* Passo induttivo ulteriore: se A€L e B€L allora AB€L

Trasformando in regole di produzione:

* S --> 01|10
* S --> 0S1 | 1S0
* S --> SS

Di conseguenza otteniamo G = ({0,1}, {S}, S, {S --> 01|10, S --> 0S1 | 1S0, S --> SS})

Esercizio 2.1 Pag 34

Determinare la grammatica che genera L = {a^n b^n | n>0} e dimostrare questo risultato. Che tipo di grammatica genera L?

X = {a, b}

L = {a^n b^n | n>0} = {ab, aabb, aaabbb, aaaabbbb, …}

Definizione di L per induzione:

* Passo base: ab€L
* Passo induttivo: Se A€L allora aAb€L

Trasformando in regole di produzione:

1. S -> ab
2. S -> aSb

Di conseguenza otteniamo G = ({a, b}, {S}, S, {S --> ab, S --> aSb}).

Per dimostrare questo risultato dobbiamo dimostrare che L = L(G), cioè che:

1. L C L(G)
2. L(G) C L

i) L(G) C L

Sia w una stringa derivabile da S (in G): w€L(G) <=> S -\*> w, w€X\*

Procediamo per induzione sulla lunghezza della derivazione di w da S. Denoto con n la lunghezza della derivazione di w da S:

* Passo base n=1. S –n(2)> ab è l’unica derivazione di lunghezza n=1 che genera stringhe di soli terminali.
* Passo induttivo. Dimostriamo che per ogni n>1, se supponiamo che il seguente enunciato è vero:

“se w’€L(G), S -n-1> w’ (w’ è derivabile in n-1 passi da S) allora w’€L”, allora anche l’enunciato: “se w€L(G), S -n> w allora w€L” risulta vero.

Consideriamo w€L(G) con S -n> w. Per definizione esiste una sequenza di forma di frase w­­1,w2,…,wn con wn=w, tale che wn deriva direttamente da S e, per ogni 1<=i<=n-1, wi+1 deriva direttamente da wi. S -> w1 -> w2 -> … -> wn = w.

E’ immediato osservare che il primo passo della derivazione è dato dall’applicazione della produzione (1) di G, allora n=1.

Si ha dunque S –(1)> aSb -n-1> wn = w.

Per ipotesi di induzione, ogni stringa derivabile da S in n-1 passi è una parola di L. Dunque, da S è possibile derivare in n-1 passi una stringa del tipo: w’ = akbk, k>0.

Più precisamente w’ = ak-1bk-1, k>0 poiché S -k(2)> akSbk, k>0.

Ma allora la stringa aw’b = aan-1bn-1b = anbn è ancora una stringa di L ed è derivabile da S in n passi. Si ha che S –(1)> aSb -n-1> aw’b = anbn = w.

Risulta dimostrato L(G) C L.

ii) L C L(G)

Sia w una parola di L. procediamo per induzione sulla lunghezza di w:

* Passo base n=1 <=> |w| = 2. w = ab. Dimostriamo che S -\*> ab. Banalmente applichiamo la produzione (2) di G e otteniamo w = ab direttamente derivabile da S. S –(2)> ab
* Passo induttivo n>1. Dimostriamo che se supponiamo l’enunciato “se w’€L, |w’| = 2(n-1) allora S -\*> w’” vero allora anche il seguente enunciato sarà vero “se w€L, |w| = 2n allora S -\*> w”.

Sia w una parola su X tale che w€L, |w| = 2n, n>1. Ovviamente w = anbn unica parola di L di lunghezza 2n. nella derivazione di w da S devo necessariamente applicare la produzione (1) di G come 1° passo. Dunque S –(1)> aSb. Per ipotesi di induzione ogni parola di L di lunghezza 2(n-1) è derivabile da S in G. Dunque anche w’ = an-1bn-1 è derivabile in S: S -\*> w’.

Ne consegue che w = anbn è derivabile da S e la relativa derivazione è ottenuta applicando in successione S –(1)> aSb e S-\*>w’, ovvero S –(1)> aSb -\*> aw’b = aan-1bn-1b = w.

Risulta dimostrato L C L(G).

Dunque L = L(G).

G è una grammatica libera da contesto perché a sinistra di ogni regola di produzione vi sono solo simboli T.

Esercizio 1 Slide Esercitazione

Sia L = {anb2n | n>0}. Determinare una grammatica corretta per L.

L = {anb2n | n>0} = {abb, aabbbb, aaabbbbbb, …}

Procediamo per induzione:

* Passo base n=1: abb€L
* Passo induttivo: se A€L allora aAbb€L

Passando alle regole di produzione:

* S –(1)> abb
* S –(2)> aSbb

In conclusione G = ({a,b}, {S}, S, {S->abb, S->aSbb}).

aaabbbbbb€L <=> S -(2)> aSbb –(2)> aaSbbbb –(1)> aaabbbbbb <=> S -\*> aaabbbbbb€L(G)

Esercizio 2 Slide Esercitazione

Sia L = {anb2n+1 | n>=0}. Determinare una grammatica corretta per L

L = {anb2n+1 | n>=0} = {b, abbb, aabbbbb, …, anbb2n}

Procediamo per induzione:

* Passo base n=0: b€L
* Passo induttivo: Se A€L allora aAbb€L

Passando alle regole di produzione S -> b|aSbb

In conclusione G = ({a,b}, {S}, S, {S->b|aSbb}).

aabbbbb€L <=> S –(2)> aSbb –(2)> aaSbbbb –(1)> aabbbbb <=> S -\*> aabbbbb€L

Esercizio 3 Slide Esercitazioni

Sia L = {anbn+mcm | n>0, m>0}. Determinare una grammatica corretta per L.

L = {anbn+mcm | n>0, m>0} = {anbnbmcm | n>0, m>0}

Determino G’ = ({a, b}, {A}, A, {A --> ab, A --> aAb}) e G’’ = ({b, c}, {B}, B, {B --> bc, B --> bBc}).

Le regole di produzione quindi diventano:

* S -> AB
* A -> ab | aAb
* B -> bc | bBc

Otteniamo G = ({a,b,c}, {S,A,B}, S, {S -> AB, A -> ab | aAb, B -> bc | bBc}).

aabbbbbccc€L ⬄ S –(1)> AB –(3)> aAbB –(2)> aabbB –(5)> aabbbBc –(5)> aabbbbBcc –(4)> aabbbbbccc <=> S -\*> aabbbbbccc€L

Esercizio 4 Slide Esercitazioni

Sia L = {anb2k+1 | n>=0, k>=0}. Determinare una grammatica corretta per L.

Procediamo per induzione su b2k+1:

* Passo base n=0: b€L’’
* Passo induttivo: se B€L allora bbB€L’’

Passando alle regole di produzione:

* B -> b
* B -> bbB

Determiniamo G’ = ({a}, {A}, A, {A-> aA|λ}) e G’’ = ({b}, {B}, B, {B->b|bbB|λ}).

V = {S,A,B}, X = {ab}, S

Definiamo le regole di produzione per G:

* S -> AB
* A -> aA| λ
* B -> b|bbB

Otteniamo G = ({a,b}, {S,A,B}, S, {S->AB, A->aA| λ, B->b|bbB})

aaabbbbb€L <=> S -(1)> AB –(2)> aAB –(2)> aaAB –(2)> aaaAB –(3)> aaaB -(5)> aaabbB –(5)> aaabbbbB –(4)> aaabbbbb <=> S -\*> aaabbbbb€L

Esercizio 5 Slide Esercitazioni

Sia L = {aibkcj | i>0, j>0, k>i+j}. Determinare una grammatica corretta per L.

L = {aibkcj | i>0, j>0, k>i+j} è simile a {aibkcj | i>0, j>0, k=i+j} ovvero {aibibzbjcj | i>0, j>0, z>0 k = i+j+z}

G’ = ({a,b}, {A}, A, {S->ab|aAb)} e G’’ = ({b,c}, {B}, B, {S->bc|bBc})

Ora determiniamo le regole di produzione su L:

* S -> ACB
* A -> ab|aAb
* B -> bc|bBc
* C -> b|bC

Otteniamo G = ({a,b}, {S,A,B,C}, S, {S -> ACB, A -> ab|aAb, B -> bc|bBc, C -> b|bC}).

abbbc€L <=> S -(1)> ACB –(2)> abCB –(4)> abBbc –(6)> abbbc <=> S -\*> abbbc€L.

Esercizio 6 Slide Esercitazione

Sia L = {a2bna1 | n>0}. Determinare una grammatica corretta per L

Procediamo per induzione:

* Passo base n=1: a2ba€L
* Passo induttivo: Se A€L allora a2bAa€L

Passando alle regole di produzione:

* S -> a2bAa
* A -> b|bA

Si ottiene G = ({a,b}, {S,A}, S, {S -> a2bAa, A -> b|bA})

Esercizio 7 Slide Esercitazione

Sia L = {anbm | n>m>0}. Determinare una grammatica corretta per L.

L = {anbm | n>m>0} = {azambm | n = m+z, z>0, m>0}

Determiniamo le regole di produzione:

* S -> AB
* A -> aA|a
* B -> ab|aBb

Otteniamo G = ({a,b}, {S,A,B}, S, {S -> AB, A -> aA|a, B -> ab|aBb})

Esercizio 8 Slide Esercitazione

Sia L = {anbmcn | n>0, m>0}. Determinare una grammatica corretta per L.

Procediamo per induzione:

* Passo base n=1: abc€L
* Passo induttivo: se A€L allora aBbc€L

Passando alle regole di produzione:

* S -> aBc | aSc
* B -> b|bB

Esercizio 2.5 Fatto da prof a lezione

L = {w€{a,b,c}\* | w = anbkc2n, n>0, k>0}. Determinare una grammatica a forma di frase o generativa o di Chomsky G corretta per L (ossia tale che L(G)=L).

L={abcc,abbcc,abbbcc,abbbbcc,aabcccc,aabbcccc,aabbbcccc,…}

S->aBcc|aScc

B->b|bB

Esercizio 2.5a da fare

L = {w€{a,b,c}\* | w =bkanc2n, n>0, k>0}. Determinare una grammatica a forma di frase o generativa o di Chomsky G corretta per L (ossia tale che L(G)=L).

Esercizio 2.5b da fare

L = {w€{a,b,c}\* | w =anc2nbk, n>0, k>0}. Determinare una grammatica a forma di frase o generativa o di Chomsky G corretta per L (ossia tale che L(G)=L).

G deve essere coerente (o consistente) AND completa. Completezza: L C L(G)

L C X\* per definizione di linguaggio formale, ma allora mi basterebbe G tale che L(G)=X\*.

Esercizio: dato un alfabeto X (ad es. X = {a,b,c}), determinare una grammatica G corretta per X\*.

Passo base: lambda€X\*

Passo induttivo: se w€X\* allora aw€X\*, bw€X\*, cw€X\*

S -> lambda|aS|bS|cS

X\* contiene tutti i linguaggi formali di alfabeto X. L formale se e solo se per def. L C X\*. G corretta per L se e solo se per def. G completa (L C L(G)) AND coerente (L(G) C L).

CAPITOLO 3

L = {w€{a,b}\* | w=anbn} -----> S->ab|aSb

L’ = {w€{a,b,c}\* | w=anbncn, n>0}

Procediamo per induzione:

* Passo base n=1: abc€L’
* Passo induttivo: se w€L’ allora awc NOT €L’ quindi S->abc|aSc non è corretta per L’

Cambiamo il passo base per ottenere un passo induttivo corretto. Vedi Esempio 3.3 Pag 70

Esercizio 1 Pag. 75

L = {w€X\* | X\* = {a,b,c}\* | w=amb3mc2n, m>0, n>0}. Determinare una grammatica libera da contesto che genera L. Dimostrare formalmente che G è corretta per L.

G = (X, V, S, P)

X = {a, b, c}

V = {S, A, B}

L = {abbb cc, abbb cccc, …}

Regole di produzione:

* S -> AB
* A -> abbb|aAbbb
* B -> cc|ccC
* B’ -> cC C-> c|cB’

B è più veloce in termine di complessità ma ha bisogno di infinita memoria.

B’ e C sono più lente ini termine di complessità ma hanno bisogno di 2 celle di memoria, quindi, può essere tramutato in algoritmo codificato in hw.

Cerca B’’ che sia più veloce di B.

G1 tale che L(G1) = {amb3m|m>0}

G2 tale che L(G2) = {c2n| m>0}

G = ({a,b,c}, {A,B,S}, S, {S -> AB, A -> abbb|aAbbb, B -> cc|ccC})

L’automa a stati finiti è una macchina che ha bisogno di n celle di memoria per computare un algoritmo. N è un numero finito.

Esercizio 2 Pag. 75

Stabilire se il seguente linguaggio è libero da contesto e giustificare formalmente la risposta:

L = {w’€{a, b, c}\* | w’ = anw | w€{b, c}\*, |w|=n, n>0}

Procediamo per induzione:

* Passo base n=1: ab€L, ac€L
* Passo induttivo: se w’€L allora aw’b€L, aw’c€L

L = {ab, ac, aabb, aabc, aacb, aacc, …}

Passando alle regole di produzione:

* S -> ab | ac| aSb | aSc

Determiniamo G = ({a, b, c}, {S}, S, { S -> ab | ac| aSb | aSc}) la quale è una grammatica corretta per L ed è libera da contesto poiché vede a sinistra delle produzioni un singolo NT. Di conseguenza L è un linguaggio libero da contesto.

CAPITOLO 4

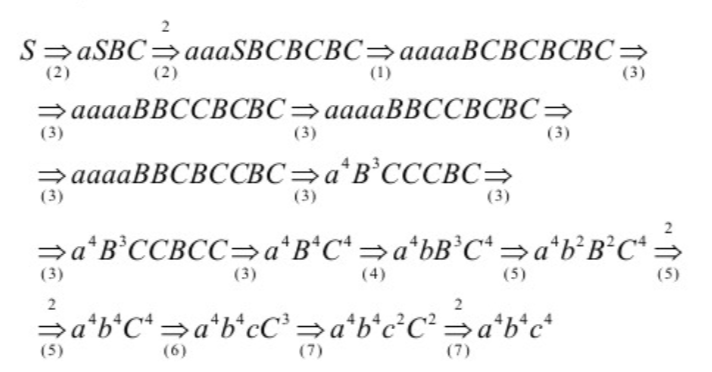
Esercizio 4.1 Pag 98

1. Determinare la grammatica che genera il linguaggio L = {anbncn | n>0} e dimostrare la correttezza di tale grammatica
2. Di che tipo è la grammatica che genera L?
3. Applicare il Pumping Lemma per dimostrare che L non è libero

1) G = (X,V,S,P) X = {a,b,c} V = {S,A,B,C} P = {S->aBC, S->aSBC, CB->BC, aB->ab, bB->bb, bC->bc, cC->cc}

Una derivazione della parola di lunghezza minima in L, w=abc, è S -1> aBC -4> abC -6> abc.

Le derivazioni sinistre delle altre parole in L sono del tipo S -2> aSBC -2> aaSBCBC. La struttura della stringa aaSBCBC non è quella desiderata. E’ necessaria una produzione che effettui uno scambio di non terminali CB ->BC. Grazie a questa regola di produzione si ha S -2> aSBC -1> aaBCBC -3> aaBBCC -4> aabBCC -5> aabbCC -6> aabbcC -7> aabbcc.

Le derivazioni di parole più lunghe in L non hanno bisogno di ulteriori produzioni: 

2) G è una grammatica monotona e per il teorema di equivalenza delle grammatiche monotone e contestuali esiste una grammatica contestuale G’ equivalente a G.

3) Supponiamo per assurdo che L sia libero da contesto. Per il Pumping Lemma sui linguaggi liberi esiste una costante p tale che: per ogni z€L, |z|>p => z=uvwxy. Inoltre, valgono le seguenti regole: (I) |vwx|<=p, (II) vx NOT= lambda, (III) per ogni i>=0: uviwxiz€L.

Consideriamo una parola in L: z=apbpcp. Il Pumping Lemma può essere applicato a tale parola poiché |z|=3p>p e dunque z può essere scritta nella forma z=uvwxy, in modo tale che |vwx|<=p.

Poiché la stringa vwx ha la lunghezza al più uguale a p, si hanno le seguenti possibilità:

1. vwx è formata da sole a, cioè, è del tipo vwx = ak, 0<k<=p;
2. vwx è formata da sole b, cioè, è del tipo vwx = bk, 0<k<=p;
3. vwx è formata da sole c, cioè, è del tipo vwx = ck, 0<k<=p;
4. vwx è a cavallo tra a e b, cioè, è del tipo vwx = akbr, 0<k+r<=p e k,r>0;
5. vwx è a cavallo tra b e c, cioè, è del tipo vwx = bkcr, 0<k+r<=p e k,r>0;

E’ immediato osservare che vwx non può essere formata da a, b e c, ossia non può essere contemporaneamente a cavallo tra a e b e tra b e c, perché non è sufficientemente lunga.

Consideriamo la stringa pompata uv2wx2y per ognuno dei casi i) – v). Per la 3) del Pumping Lemma sui linguaggi liberi si dovrebbe avere che uv2wx2y€L.

Ma nel caso:

1. aggiungiamo almeno una a, ed al più p a; uv2wx2y = ap+tbpcp NOT€L, 0<t<=p;
2. aggiungiamo almeno una b, ed al più p b; uv2wx2y = apbp+tcp NOT€L, 0<t<=p;
3. aggiungiamo almeno una c, ed al più p c; uv2wx2y = apbpcp+t NOT€L, 0<t<=p;
4. per la 2) del Pumping Lemma, si hanno le seguenti possibiità:
   1. v NOT= lambda, x NOT= lambda
   2. v NOT= lambda, x = lambda
   3. v = lambda, x NOT= lambda

Osserviamo preliminarmente che, se v NOT= lambda, allora v è costituita da sole a. Infatti, se v fosse del tipo v = akbr’, con 0<r’<=r, si avrebbe uv2wx2y = ap-kakbr’akbr’bscp NOT€L, con p-r’<=s<=2(r-r’)+p-r. Analogamente, se x NOT= lambda, allora x è costituita da sole b. Infatti, se x fosse del tipo x = ak’br, con 0<k’<=k, si avrebbe uv2wx2y = asak’brak’brbp-rcp NOT€L, con p-k’<=s<=2(k-k’)+p-k.

Per cui:

1. se v NOT=lambda e x NOT= lambda per l’osservazione precedente v = ak’, con 0<k’<=k, e x = br’, con 0<r’<=r, e si ha che uv2wx2y = ap+k’bp+r’cp NOT€L poiché k’, r’>0;
2. se v NOT=lambda e x = lambda si ha v = ak’, con 0<k’<=k, e uv2wx2y = ap+k’bpcp NOT€L poiché k’>0;
3. se v =lambda e x NOT= lambda si ha x = br’, con 0<r’<=r, e uv2wx2y = apbp+r’cp NOT€L poiché r’>0.
4. per la 2) del Pumping Lemma, si hanno le seguenti possibiità:
   1. v NOT= lambda, x NOT= lambda
   2. v NOT= lambda, x = lambda
   3. v = lambda, x NOT= lambda

Osserviamo preliminarmente che, se v NOT= lambda, allora v è costituita da sole b. Infatti, se v fosse del tipo v = bkcr’, con 0<r’<=r, si avrebbe uv2wx2y = apbp-kbkcr’bkcr’cs NOT€L, con p-r’<=s<=2(r-r’)+p-r. Analogamente, se x NOT= lambda, allora x è costituita da sole c. Infatti, se x fosse del tipo x =bk’cr, con 0<k’<=k, si avrebbe uv2wx2y = apbsbk’crbk’crcp-r NOT€L, con p-k’<=s<=2(k-k’)+p-k.

Per cui:

1. se v NOT=lambda e x NOT= lambda per l’osservazione precedente v = bk’, con 0<k’<=k, e x = cr’, con 0<r’<=r, e si ha che uv2wx2y = apbp+k’cp+r’ NOT€L poiché k’, r’>0;
2. se v NOT=lambda e x = lambda si ha v = bk’, con 0<k’<=k, e uv2wx2y = apbp+k’cp NOT€L poiché k’>0;
3. se v =lambda e x NOT= lambda si ha x = cr’, con 0<r’<=r, e uv2wx2y = apbpcp+r’ NOT€L poiché r’>0.

In ciascuno dei casi i) – v) uv2wx2y NOT€L. Assurdo!

Ne segue che L non è un linguaggio libero da contesto, poiché è un linguaggio infinito per il quale non vale il Pumping Lemma.

Esercizio 4.2 Pag 103

Applicare il Pumping Lemma per dimostrare che L non è libero da contesto: L = {an^2 | n>=0}.

Analizziamo le parole che costituiscono L. L = {lambda, a, a4, a9, a16, …}

Supponiamo per assurdo che L sia libero da contesto. Per il Pumping Lemma sui linguaggi liberi, esiste un numero naturale p, dipendente solo da L, tale che, se z€L e |z|>p, allora z=uvwxy e quindi:

1. |vwx|<=p
2. vx NOT= kambda
3. uviwxiy € L per ogni i>=0

Consideriamo la parola z=ap^2. z€L ed inoltre |z|=p2>p

Per il Pumping Lemma possiamo scrivere z=uvwxy ove |vwx|<=p. Consideriamo la stringa uv2wx2y. Per la (3) del Pumpinig Lemma si deve avere che uv2wx2y €L.

Ma |uv2wx2y|=|uvwxy|+|vx| <= |uvwxy|+|vwx|<= p2+p < p2 + 2p + 1 = (p+1)2

(|uv2wx2y|>|uvwxy| = |z| per la (2) del pumping lemma, il che implica che uv2wx2y NOT= z)

(p+1)2 = |a(p+1)^2| e uv2wx2y NOT= a(p+1)^2 perché |uv2wx2y|<(p+1)2 = |a(p+1)^2|

Dunque, uv2wx2y NOT€ L, assurdo. Ne segue che L non è un linguaggio libero da contesto per p>1

Se p=1, z=a(p+1)^2 e |z|=(p+1)2 > p, assurdo.

Esercizio 4.3 Pag 104

Applicare il Pumping Lemma per dimostrare che il seguente linguaggio L non è libero da contesto:

L = {aibj | i=2j, i>=0, j>=0}

Analizziamo le parole che costituiscono L: L = {aibj | i=2j, i>=0, j>=0} = {a,a2b, a4b2, a8b3, a16b4, …}

Supponiamo per assurdo che il sia libero da contesto. Vale dunque per L il Pumping Lemma sui linguaggi liberi, ovvero esiste un numero naturale p, dipendente solo da L, tale che, se z€L e |z|>p, allora z=uvwxy e quindi:

1. |vwx|<=p
2. vx NOT= kambda
3. uviwxiy € L per ogni i>=0

Consideriamo la parola z = a2^pbp. z€L e |z|= 2p+p > p

Per il pumping lemma possiamo scrivere z = uvwxy ove |vwx|<=p per la (1) del PL. Consideriamo ora la stringa uv2wx2y. Per la (3) del Pumpinig Lemma si deve avere che uv2wx2y €L.

Ma |uv2wx2y| = |uvwxy|+|vw| <= |uvwxy|+|vwx| <= 2p + 2p + p = 2\*2p + p = 2p+1 + p < 2p+1 + p + 1.  (|uv2wx2y|>|uvwxy| = |z| per la (2) del pumping lemma, il che implica che uv2wx2y NOT= z perché 2p+1+p+1 = |a2^p+1bp| e uv2wx2y NOT= a2^p+1bp perché |uv2wx2y|<2p+1+p+1 = |a2^p+1bp|)

Dunque, la stringa pompata uv2wx2y non è del tipo aibj, ossia uv2wx2y NOT€ L, assurdo. Ne segue che L non è un linguaggio libero da contesto.

Esercizio 4.4 Pag 104

Dimostrare che il seguente linguaggio L non è libero: L = {akbr | k>0, r>k2}

Analizziamo le parole che costituiscono L: L = {akbr | k>0, r>k2} = {ab2, ab3,ab4,…,a2b5,a2b6,a2b7}

Supponiamo per assurdo che L sia libero da contesto. Vale dunque per L il Pumping Lemma sui linguaggi liberi, ovvero esiste un numero naturale p, dipendente solo da L, tale che, se z€L e |z|>p, allora z=uvwxy e quindi:

1. |vwx|<=p
2. vx NOT= kambda
3. uviwxiy € L per ogni i>=0

Consideriamo la parola z=apbp^2+1. Z€L e |z|=p+p2+1 > p

Per il pumping lemma possiamo scrivere z = uvwxy. Per la sottostringa vwx si hanno le seguenti possibilità:

1. vwx formata da sole a: vwx=ak 0<k<=p
2. vwx formata da sole b: vwx=bk 0<k<=p
3. vwx a cavallo tra a e b: vwx=atbs 0<t+s<=p t>0 s>0

Nel 1) caso per la (1) e la (2) del PL si ha che 0<|vwx|<=p e 0<|vx|<=p. dunque vwx può essere formata da almeno una a ed al più p a. Consideriamo la parola uv2wx2y: tale parola ha almeno una ed al più p a in più di z. dunque, uv2wx2y ha almeno p+1 a ed al più 2p a, mentre il numero delle b non è mutato. Ossia, denominato con #(x) il numero di occorrenze del simbolo x in uv2wx2y: p+1 <= #(a) <= 2p e #(b) = p2+1. Ma questo vuol dire che #(b) = p2+1 < (p+1)2 <= #(a)2 <= 4p2. Dunque, r<k2, assurdo quindi uv2wx2y€L

Nel 2) caso, vwx è formata da almeno una p e al più p b. Consideriamo la parola uv0wx0y. Tale parola ha almeno una e al più p b in meno di z, in quanto per la (1) e la (2) del PL 0<|uvx|<=p. dunque uv0wx0y ha al più p2-1 b ed almeno p2-1-p b, mentre il numero di a non è cambiato. Quindi si ha che p2-p-1<=#(b)<=p2-1 e #(a) = p. Ma questo vuol dire #(b)<=#(a)2=p2, dunque si ha che r<=k2 il che è un assurdo che porta uv0wx0y NOT€ L.

Nel 3) caso, vwx è formata sua da a che da b e 0<#(a)+#(b)<=p. Poiché per la (1) e la (2) del PL 0<|vwx|<=p, se v NOT= lambda, allora v contiene solo a, altrimenti vi, i>1 conterebbe delle a alternate a delle b. analogamente, se x NOT= lambda allora x contiene solo b. dunque ci sono tre possibilità:

3a) v NOT= lambda e x = lambda

3b) v = lambda e x NOT= lambda

3c) v NOT= lambda NOT= x

Nel caso 3a consideriamo la parola uv2wx2y. Come nel caso 1) tale parola ha almeno una a in più rispetto a z ed al più p-1 a in più di z poiché ci deve essere almeno una b in vwx e questa deve essere necessariamente in w. Il numero delle b in uv2wx2y non è mutato. Si ha che p+1 <= #(a) <= p+p-1 = 2p-1 e #(b)=p2+1 e quindi poiché #(b)=p2+1 < (p+1)2 <= #(a) <= (2p-1)2 si ha che r<k2, il che è un assurdo che porta uv2wx2y NOT€ L.

Nel caso 3b consideriamo la parola uv0wx0y. Come nel caso 2) tale parola ha almeno una b in meno rispetto a z ed al più p-1 b in meno di z poiché ci deve essere almeno una a in vwx e questa deve necessariamente essere in w. Il numero delle a in uv0wx0y non è mutato. Si ha che p2+1-p+1 = p2- p+2 <= #(b) <= p2 e #(a)=p e quindi #(b)<=#(a)2=p2, dunque r<k2 il che è un assurdo che porta uv0wx0y NOT€L.

Nel caso 3c consideriamo la parola uv2wx2y. Poiché v NOT= lambda, v contiene almeno una a. Dunque, v2 contiene almeno due a e uv2wx2y contiene almeno p+1 a. Ma |uv2wx2y|=|uvwxy|+|vx|=|z|+|vx|<=|z|+|vwx|<= p+p2+1+p = p2+2p+1 = (p+1)2 < p+2+(p+1)2. Dunque, uv2wx2y ha almeno p+1 a, ma la sua lunghezza è strettamente minore della lunghezza della parola di L con almeno p+1 a e di lunghezza di minima. Ne consegue che uv2wx2y NOT€ L.

In ciascuno dei tre casi e sottocasi risulta violata la (3) del PL per i linguaggi liberi che è un assurdo. L’assurdo deriva dall’aver assunto L libero di contesto. Dunque, L non è un linguaggio libero.

Esercizio

Stabilire se il seguente linguaggio L è libero. Giustificare formalmente la risposta. L = {w€X\*|w=anbm, m>n>0}.

Analizziamo le parole che caratterizzano L. L = {ab2,ab3ab4,…,a2b3,a2b4,a2b5,…,a3b4,a3b5,a3b6,…}

m>n -> m=n+k, k>0 -------> L={w€X\*|w=anbn+k, n>0, k>0}={ w€X\*|w=anbnbk, n>0, k>0}

S->AB

A->ab|aAb

B->b|bB

G=({a,b}, {S,A,B}, S, {S->AB, A->ab|aAb, B->b|bB})

L è un linguaggio libero da contesto poiché esiste una grammatica G libera che genera L.

Esercizio

Stabilire se il seguente linguaggio L è libero. Giustificare formalmente la risposta. L = {w€X\*|w=bn, n>0}.

Analizziamo le parole che caratterizzano L. L={b,bb,bbb,bbbb,bbbbb,bbbbbb,…}

S->bS|b

G=({b}, {S,B}, S, {S->bS})

L è un linguaggio libero da contesto poiché esiste una grammatica G libera che genera L.

Esercizio

Stabilire se il seguente linguaggio L è libero. Giustificare formalmente la risposta. L = {w€X\*|w=anbmcn, m>=n>0 = {abc,abbc,abbbc,abbbbc,…,aabbcc,aabbbcc,aabbbbcc, …}

Dimostrare che L non è CF utilizzando il Teorema uvwxy.

Supponiamo per assurdo che L sia libero da contesto. Per il Pumping Lemma sui linguaggi liberi esiste una costante p tale che per ogni z€L |z|>p ---> z=uvwxy. Inoltre, valgono le seguenti regole:

1. |vwx|<=p
2. vx NOT= kambda
3. uviwxiy € L per ogni i>=0

Consideriamo z€L, z=apbpcp, |z|=3p>p. Il PL può essere applicato a z che può essere riscritta come z=uvwxy, in modo che |vwx|<=p. Poiché la lunghezza massima di |vwx| è p, si hanno le seguenti possibilità:

1. vwx è formata da sole a, cioè, è del tipo vwx = ak, 1<=k<=p;
2. vwx è formata da sole b, cioè, è del tipo vwx = bk, 1<=k<=p;
3. vwx è formata da sole c, cioè, è del tipo vwx = ck, 1<=k<=p;
4. vwx è a cavallo tra a e b, cioè, è del tipo vwx = atbs, 1<=t<=p, 1<=s<=p e t,s>0;
5. vwx è a cavallo tra b e c, cioè, è del tipo vwx = btcs, 1<=t<=p, 1<=s<=p e t,s>0

VINCOLI:

1. #(b)>=#(a)
2. #(b)>=#(c)
3. #(a)=#(c)

Nel caso i) per la (1) e la (2) del PL si ha che 0<|vwx|<=p e 0<|vw|<=p. Dunque vwx è formata da almeno una a e al più p a in più rispetto a z. Consideriamo la parola pompata uv2wx2y. Questa parola ha almeno una ed al più p a in più rispetto a z. Dunque uv2wx2y ha almeno p+1 a e al più 2p a, mentre il numero delle b e delle c non è mutato. Quindi, denotato #(x) il numero di occorrenze di x in uv2wx2y, abbiamo che p+1<=#(a)<=2p, #(c)=p, #(b)=p. Questo viola i vincoli 1 e 3, di conseguenza uv2wx2y NOT€L.

Nel caso ii) per la (1) e la (2) del PL si ha che 0<|vwx|<=p e 0<|vw|<=p. Dunque vwx è formata da almeno una b e al più p b in meno rispetto a z. Consideriamo la parola depompata uv0wx0y. Questa parola ha almeno una ed al più p b in meno rispetto a z. Dunque uv0wx0y ha almeno 0 b e al più p-1 b, mentre il numero delle a e delle c non è mutato. Quindi, denotato #(x) il numero di occorrenze di x in uv2wx2y, abbiamo che 0<=#(b)<=p-1, #(c)=p, #(a)=p. Questo viola i vincoli 1 e 2, di conseguenza uv2wx2y NOT€L.

Nel caso iii) per la (1) e la (2) del PL si ha che 0<|vwx|<=p e 0<|vw|<=p. Dunque vwx è formata da almeno una c e al più p c in più rispetto a z. Consideriamo la parola pompata uv2wx2y. Questa parola ha almeno una ed al più p c in più rispetto a z. Dunque uv2wx2y ha almeno p+1 c e al più 2p c, mentre il numero delle b e delle a non è mutato. Quindi, denotato #(x) il numero di occorrenze di x in uv2wx2y, abbiamo che p+1<=#(c)<=2p, #(a)=p, #(b)=p. Questo viola i vincoli 2 e 3, di conseguenza uv2wx2y NOT€L.

Nel caso iv) vwx è formata sia da a che da b e 0<#(a)+#(b)<=p. Per la (1) e la (2) del PL 0<|vwx|<=p. Se v NOT= lambda, allora v contiene solo a (altrimenti vi, i>1, conterrebbe delle a alternate a delle b). Se x NOT= lambda, allora x contiene solo b. Si verificano tre possibilità:

1. v NOT= lambda e x = lambda
2. v = lambda e x NOT= lambda
3. v NOT= lambda NOT= x

Nel caso a) consideriamo la parola uv2wx2y. Come nel caso 1), tale parola ha almeno una a e al più p-1 a in più di z poiché ci deve essere almeno una b in vwx e questa deve essere necessariamente in w. Il numero delle b non è mutato, si ha che p+1 <= #(a) <= 2p-1 e #(b) = p e quindi, poiché #(b)=p < #(a) = p+1 si ha che uv2wx2y NOT€L.

Nel caso b) consideriamo la parla uv0wx0y. Come nel caso 2), tale parola ha almeno una b e al più p-1 b in meno rispetto a z poiché ci deve essere almeno una a in vwx e questa deve essere necessariamente in w. Il numero delle a non è mutato, si ha che p – (p-1) = 1 <= #(b) <= p-1 e #(a)=p e quindi, poichè #(b)<#(a)=p si ha che uv0wx0y NOT€L.

Nel caso c) consideriamo la parola uv2wx2y. Poiché v NOT= lambda, v contiene almeno una a. Dunque v2 contiene almeno due a e uv2wx2y contiene almeno p+1 a. Ma |uv2wx2y|=|uvwxy|+|vx|=|z|+|vx|<=|z|+|vwx| <= p+p+p+p = 4p < p+1+p+1+p+1+p+1 = 4p+4. Il che è un assurdo, di conseguenza uv2wx2y NOT€L.

Nel caso v) vwx è formata sia da b che da c e 0<#(b)+#(c)<=p. Per la (1) e la (2) del PL 0<|vwx|<=p. Se v NOT= lambda, allora v contiene solo b (altrimenti vi, i>1, conterrebbe delle b alternate a delle c). Se x NOT= lambda, allora x contiene solo c. Si verificano tre possibilità:

1. v NOT= lambda e x = lambda
2. v = lambda e x NOT= lambda
3. v NOT= lambda NOT= x

Nel caso a) consideriamo la parla uv0wx0y. Come nel caso 2), tale parola ha almeno una b e al più p-1 b in meno rispetto a z poiché ci deve essere almeno una c in vwx e questa deve essere necessariamente in w. Il numero delle c non è mutato, si ha che p – (p-1) = 1 <= #(b) <= p-1 e #(c)=p e quindi, poichè #(b)<#(c)=p si ha che uv0wx0y NOT€L.

Nel caso b) consideriamo la parola uv2wx2y. Come nel caso 1), tale parola ha almeno una c e al più p-1 c in più di z poiché ci deve essere almeno una b in vwx e questa deve essere necessariamente in w. Il numero delle b non è mutato, si ha che p+1 <= #(c) <= 2p-1 e #(b) = p e quindi, poiché #(b)=p < #(c) = p+1 si ha che uv2wx2y NOT€L.

Nel caso c) consideriamo la parola uv2wx2y. Poiché x NOT= lambda, x contiene almeno una c. Dunque x2 contiene almeno due c e uv2wx2y contiene almeno p+1 c. Ma |uv2wx2y|=|uvwxy|+|vx|=|z|+|vx|<=|z|+|vwx| <= p+p+p+p = 4p < p+1+p+1+p+1+p+1 = 4p+4. Il che è un assurdo, di conseguenza uv2wx2y NOT€L.

In ciascuno dei cinque casi e sottocasi risulta violata la (3) del PL per i linguaggi liberi che è un assurdo. L’assurdo deriva dall’aver assunto L libero di contesto. Dunque, L non è un linguaggio libero.

Esercizio

Stabilire se il seguente linguaggio L è libero. Giustificare formalmente la risposta. L = {w€X\*|w=aibkcj, k=i+j, i>0, j>0} = {aibibjcj, i>0, j>0} = {abbc, abbbcc,aaabbbbc, …}

S->AC

A->ab|aAb

C->bc|bCc

G=({a,b,c}, {S,A,B}, S, {S->AC, A->ab|aAb, C->bc|bCc})

L è un linguaggio libero da contesto poiché esiste una grammatica G libera che genera L.

Esercizio

Stabilire se il seguente linguaggio L è libero. Giustificare formalmente la risposta. L = {w€X\*|w=akbicj, k=i+j, i>0, j>0} = {ajaibicj, i>0, j>0} = {abbc, abbbcc,aaabbbbc, …}

S->aSc|aS1c

S1->aS1b|ab

G=({a,b,c}, {S,S1}, S, {S->aSc|aS1c, S1->aS1b|ab})

L è un linguaggio libero da contesto poiché esiste una grammatica G libera che genera L.

Esercizio

Stabilire se L è libero e giustificare formalmente la risposta. L = {w€X\* | w=at, t=n3, n>0} = {a,a8,a27,a64,…}.

L non è un linguaggio libero da contesto poiché la lunghezza delle parole di L non cresce in maniera costante e non è possibile determinare un sottoinsieme infinito di L le cui parole hanno lunghezze che crescono in maniera costante.

Per provarlo formalmente, supponiamo che L sia libero da contesto. Per il Pumping Lemma sui linguaggi liberi deve esistere una costante intera p, dipendente unicamente da L, tale che per ogni z€L |z|>p: z=uvwxy ed inoltre valgono le seguenti proprietà:

1. |vwx|<=p
2. vx NOT= lambda
3. Per ogni i>0 uviwxiy € L

Consideriamo la parola z€L z=a(p+1)^3 dove |z|=(p+1)3>p. Il PL può essere applicato a z che può essere riscritta come z=uvwxy, in modo che |vwx|<=p per la (1) del PL. Consideriamo ora la stringa pompata uv2wx2y. Per la (3) del Pumpinig Lemma si deve avere che uv2wx2y €L.

Ma |uv2wx2y| = |z|+|vw| <= |z|+|vwx| <= (p+1)3 + p < (p+2)3 = |a(p+2)^3| (|uv2wx2y|>|uvwxy| = |z| per la (2) del pumping lemma, il che implica che uv2wx2y NOT= z perché (p+2)3= |a(p+2)^3| e uv2wx2y NOT= a(p+2)^3 perché |uv2wx2y|< (p+2)3 = |a(p+2)^3|).

Dunque, la stringa pompata uv2wx2y non è del tipo at, ossia uv2wx2y NOT€ L, assurdo. Ne segue che L non è un linguaggio libero da contesto.

Esercizio

Stabilire se L è libero e giustificare formalmente la risposta. L = {w€X\* | aibj, 0<i<=j<=2i}

Analizziamo le parole che costituiscono L: L = {w€X\* | aibj, 0<i<=j<=2i} = {ab, ab2,a2b2,a2b3,a2b4,a3b3,…}

Passo base: ab€L, abb€L

Passo induttivo: Se w€L allora awb€L e awbb€L

S->ab|abb|aSb|aSbb|

Esercizio

Stabilire se L è libero e giustificare formalmente la risposta. L = {w€X\* | aibj, 0<i<j<2i}

Analizziamo le parole che costituiscono L: L = {w€X\* | aibj, 0<i<j<2i} = {a2b3,a3b4,a3b5,a4b5,…}

Passo base: aabbb€L

Passo induttivo: se w€L allora awb€L e awbb€L

S->aabbb|aSb|aSbb

Esercizio

Stabilire se L è libero e giustificare formalmente la risposta. L = X\* = {lambda, 0, 1, 00,01,10,11,…}

Passo base: lambda€L

Passo induttivo: se w€L allora 0w€L e 1w€L

S->0S|1S|lambda

G=({lamda,0,1}, {S}, S, { S->0S|1S|lambda})

Se X fosse ternario S->lambda|0S|1S|2S

Se X fosse decimale S->lambda|0S|1S|2S|3S|4S|5S|6S|7S|8S|9S

Quindi per generare X\* abbiamo bisogno di |X|+1 produzioni

Esercizio

Stabilire se L è libero e giustificare formalmente la risposta. L = {w€X\* | aibj, 0<=j<=i}

i=j+k k>=0 --->{w€X\* | akajbj, j>=0, k>=0}

S->S1S2

S1->lambda|aS1

S2->lambda|aS2b

Esercizio

Stabilire se L è libero e giustificare formalmente la risposta. L = X+, X={a}

S->a|aS

Esercizio

Stabilire se L è libero e giustificare formalmente la risposta. L = {w€X\* | w=aibkcj, k=2i or k=j, i>0,j>0} = {w€X\* | w=aib2icj,i>0,j>0} U {w€X\* | w=aibjcj, i>0,j>0}

S1->S1’S1’’

S1’->abb|aS1’bb

S1’’->c|cS1’’

S2->S2’S2’’

S2’->a|aS2’

S2’’->bc|bS2’’c

S->S1S2

Esercizio

Stabilire se L è libero e giustificare formalmente la risposta. L = {w€X\* | w=aibjck, 0<i+j<k, i>0, j>0} = {abccc,abcccc,abc5,…,a2bc4,a2bc5,…,ab2c4,ab2c5,…} = {w€X\* | w=aibjci+j+n, i,j,n>0} = {w€X\* | w=aibjcicjcn, i,j,n>0} = = {w€X\* | w=aibjcjcicn, i,j,n>0}

S->S1S2

S1->aS1c|aS’c

S’->bS’c|bc

S2->c|cS2

Esercizio

Stabilire se L è libero e giustificare formalmente la risposta. L = {w€X\* | w=aibjak, 0<i+j<k, i>0,j>0} = {w€X\* | w=aibjai+j+n, i,j,n>0} = {w€X\* | w=aibjaiajan, i,j,n>0} = = {w€X\* | w=aibjajaian, i,j,n>0}

S->S1S2

S1->aS1a|aS’a

S’->bS’a|ba

S2->a|aS2

Capitolo 6

CAPITOLO 7

Esempio 7.1 Pag 184

R = ba\*

Determinare una grammatica di tipo 3 G corretta per il linguaggio denotato dall’espressione regolare R, ossia tale che L(G)=S(R). Costruire un FSA M con funzione delta totale che riconosce il linguaggio denotato dall’espressione regolare R, ossia tale che T(M)=S(R).

S(R) = S(ba\*)= S(B) S(a\*) = S(B) (S(a))\* = {b}{a}\* = {w€X\* | w=ban, n>=0} con X={a,b }

S(R) può essere visto come L=L1 L2 con L1={b} e L2={a\*}. Quindi G1 = ({b},{S}, S, {S-->b}) e G2=({a}, {S}, S, {S-->lambda|aS}). Di conseguenza G = ({a,b}, {S,S1}, S, {S-->bS1, S1-->lambda, aS1}).  
  
Costruiamo ora un FSA deterministico M con funzione totale δ tale che L=T(M).

M=(Q, δ, q0, F) ove Q = Vu{q0}, δ: QxX-->Q, q0€V, F={q0}u{S1}

δ (S,b)=S1 S1€ δ(S1,a)

Esercizio 7.1 pag 201

Determinare una grammatica lineare destra che genera il linguaggio descritto dalla seguente espressione regolare: b\*+(ab)\*.  
Il linguaggio relativo all’espressione regolare è: S(b\*+(ab)\*) = S(b\*) U (S(ab))\* = {b}\* U (S(a)S(b))\* = {b}\* U ({a}{b})\* = {b}\* U {ab}\*

Costruiamo la grammatica G1 tale che L(G1)=S(b\*)={b}\* ---> G1=(X1, V1, S1, P1) = ({b},{S1},S1,{S1-> λ|bS1})

Costruiamo la grammatica G2 tale che L(G2)=(S(ab))\*={ab} --->G2=(X2, V2, S2, P2) = ({a,b}, {S2, B2}, S2, {S2-->aB2, B2-->b})

Costruiamo la grammatica G3 tale che L(G3)=S((a,b)\*)={ab}\* applicando il teorema di chiusura sull’iterazione su linguaggi di tipo 3. G3=(X3,V3,S3,P3) con:

* X3 = X2 = {a,b}
* V3 = S3UV2={S3,S2}
* P3 = {S3--> λ} U (P2-{S--> λ}) U {S3-->w|S2-->€P2} U {A-->bS3|A-->b€P2} = {S3--> λ} U P2 U {S3-->aB2} U {B2-->bS3} = {S3--> λ|aB2, S2-->aB2, B2-->b|bS3}

Dato che S2 non compare come non terminale a destra di nessuna produzione in P3, S2 diventa inutile e quindi tutte le sue produzioni possono essere rimosse. Quindi P3 diventa {S3--> λ|aB2, B2-->b|bS3}

Ora possiamo costruire la grammatica G tale che L(G)=S(b\*+(ab)\*)={b}\*U{ab}\*=G1UG3. Per il teorema di chiusura dell’unione sui linguaggi di tipo 3 G=(X,V,S,P) tale che:

* X = X1 U X3 = {a,b}
* V = V1 U V3 = V1 U V2 U {S3} U {S} = {S,S1,B2,S3}
* P = {S-->w|S1-->w€P1} U {S-->w|S3-->w€P3} U P1 U P3 = {S-> λ|bS1} U {S--> λ|aB2} U {S1-> λ|bS1} U {S3--> λ|aB2, B2-->b|bS3} = {S-> λ|bS1|aB2, S1-->bS1|λ, S3--> λ|aB2, B2-->b|bS3}

Data G grammatica di tipo 3, individuare un FSA M tale che L(G)=T(M).

Per il teorema di Kleen Lgotico3 = LgoticoFSL = LgoticoREG. In particolare, utilizziamo la dimostrazione di Lgotico3 C LgoticoFSL. Esiste almeno un NDA che riconosce L. Possiamo costruire M = (Q,δ,q0,F) tale che:

* Q = V U q = {S,S1,B2,S3, q}
* q0 = S
* F = {q} U {S, S1, S3}
* δ: QxX-->2^Q definita:
  + S-->aB2 B2€δ(S,a)
  + S-->bS1 S1€δ(S,b)
  + S1-->bS1 S1€δ(S1,b)
  + S3-->aB2 B2€ δ(S3,a)
  + B2-->bS3 S3€ δ(B2,b)
  + B2-->b q€δ(B2,b)

Otteniamo un NDA perché partendo da B2 otteniamo un insieme di stati dalla funzione di transizione. Possiamo costruire un FSA M’ equivalente ad M applicando l’algoritmo di trasformazione da NDA a FSA:

M’ = (Q’, δ’, q’0, F’) con alfabeto d’ingresso X ove:

* Q’ = 2Q = 2{S,S1,S3,B2,q}
* q’0 = {S}
* F’ = {{q}, {S}, {S1}, {q,S3}}
* δ’: QxX:Q definita:
  + δ’({S},a) = δ(S,a) = {B2}
  + δ’({S},b) = δ(S,b) = {S1}
  + δ’({S1},a) = δ(S1,a) = ∅
  + δ’({S1},b) = δ(S1,b) = {S1}
  + δ’({S3},a) = δ(S3,a) = {B2}
  + δ’({S3},b) = δ(S3,b) = ∅
  + δ’({B2},a) = δ(B2,a) = ∅
  + δ’({B2},b) = δ(B2,b) = {q,S3}
  + δ’({q,S3},a) = δ(q,a) U δ(S3,a) = ∅ U {B2} = {B2}
  + δ’({q,S3},b) = δ(q,b) U δ(S3,b) = ∅

Esercizio 7.2 Pag 203

Data la seguente grammatica lineare destra G=(X,V,S,P) con:

* X = {a,b,c}
* V = {S,A,B}
* P = {S->bA|aS|b(1), A->aB|cS|a(2), B->bA|cB|c(3)}

Determinare un’espressione regolare che denota il linguaggio L(G). Passiamo da produzioni a insiemi di stringhe derivabili in G dai NT A, B ed S.

A = {a}B U {c}S U {a} = aB U cS U a (2’)

Sostituendo la (2’) nella (1) otteniamo S = b(aB U cS U a) U aS U b = baB U bcS U ba U aS U b = baB U (bc U a)S U (ba U b) e si ottiene (1’)

Sostituendo in (3) la (2’) otteniamo B = baB U bcS U ba U cB U c = (ba U c)B U (bcS U c U ba) e si ottiene (3’)

Denotiamo con A,B ed S tre espressioni regolari e otteniamo (1’) = S = baB + (bc+a)S + (ba+b), mentre (3’) = B = (ba+c)B + (bcS+ba+c)

In entrambi i casi riconosciamo R1=R2R1 + R3 con R2 != λ. Per la proprietà 20) sulle espressioni regolari si ha R1 = R2\*R3.

Dunque otteniamo (3’’) = B = (ba+c)\*(bcS+ba+c)

Sostituiamo (3’’) in (1’) si ottiene S = ba(ba+c)\*(bcS+ba+c) + (bc + a)S + (ba + b) = ba(ba+c)\*bcS + (a+bc)S + ba(ba+c)\*(ba+c) + ba + b = (bc+a+ba(ba+c)\*bc)S + ba(ba+c)\*(ba+c)+ba+b = per la proprietà 20) sulle espressioni regolari (ba(ba+c)\*bc+a+bc)\*(ba(ba+c)\*(ba+c)+ba+b)

Data G grammatica di tipo 3, individuare un FSA M tale che L(G)=T(M).

Per il teorema di Kleen Lgotico3 = LgoticoFSL = LgoticoREG. In particolare, utilizziamo la dimostrazione di Lgotico3 C LgoticoFSL. Esiste almeno un NDA che riconosce L. Possiamo costruire M = (Q,δ,q0,F) tale che:

* Q = V U q = {S,A,B,q}
* q0 = S
* F = {q} U {S,A,B}
* δ: QxX-->2^Q definita:
  + S € δ(S,a)
  + A € δ(S,b)
  + q € δ(S,b)
  + ∅ € δ(S,c)
  + B δ (A,a)
  + q € δ(A,a)
  + ∅ € δ(A,b)
  + S € δ(A,c)
  + ∅ € δ(B,a)
  + A € δ(B,b)
  + B € δ(B,c)
  + q € δ(B,c)

Otteniamo un NDA perché partendo da S, oppure da A, oppure da B otteniamo un insieme di stati dalla funzione di transizione. Possiamo costruire un FSA M’ equivalente ad M applicando l’algoritmo di trasformazione da NDA a FSA:

M’ = (Q’, δ’, q’0, F’) con alfabeto d’ingresso X ove:

* Q’ = 2Q = 2{S, A, B, q}
* q’0 = {S}
* F’ = {{q}, {A,q}, {B,q}}
* δ’: QxX:Q definita:
  + δ’({S},a) = δ(S,a) = {S}
  + δ’({S},b) = δ(S,b) = {A,q}
  + δ’({S},c) = δ(S,c) = ∅
  + δ’({A},a) = δ(A,a) = {B,q}
  + δ’({A},b) = δ(A,b) = ∅
  + δ’({A},c) = δ(A,c) = {S}
  + δ’({B},a) = δ(B,a) = ∅
  + δ’({B},b) = δ(B,b) = {A}
  + δ’({B},c) = δ(B,c) = {B,q}
  + δ’({A,q},a) = δ(A,a) U δ (q,a) = {B,q} U ∅ = {B,q}
  + δ’({A,q},b) = δ(A,b) U δ (q,b) = ∅
  + δ’({A,q},c) = δ(A,c) U δ (q,c) = {S} U ∅ = {S}
  + δ’({B,q},a) = δ(B,a) U δ (q,a) = ∅
  + δ’({B,q},b) = δ(B,b) U δ (q,b) = {A} U ∅ = {A}
  + δ’({B,q},c) = δ(B,c) U δ (q,c)= {B,q} U ∅ = {B,q}

Esercizio 7.3 Pag 205

Sia L il linguaggio denotato dall’espressione regolare (aa+aaa)\*

1. Trovare l’automa a stati finiti che riconosce L
2. Trasformare l’NDA del punto 1 in un automa FSA equivalente.

1) Determiniamo G1 e G2 grammatiche lineari destre tali che L(G1)=S{aa}={aa} e L(G2)=S(aaa)={aaa} entrambe con X={a}.

G1 = (X,V1,S1,P1) con V={S1,A} e P={S1->aA, A->a}

G2 = (X,V2,S2,P2) con V={S2,B,C} e P={S2->aB, B->aC, C->a}

Determiniamo ora la grammatica G3 lineare destra tale che L(G3)=L(G1)UL(G2)=S(aa)US(aaa)=S(aa+aaa)

G3 = (X,V3,S3,P3) ove V3=V1UV2U{S3}={S1,S2.S3,A,B,C} e P3 = {S3->aA|aB, S1->aA, A->a, S2->aB, B->aC, C->a} = {S3->aA|aB, A->a, B->aC, C->a}

Determiniamo ora la grammatica G lineare destra tale che L(G)=(L(G3)\*)=S((aa+aaa)\*)

G = (X,V,S,P) con V=V3US = {S,S1,S2.S3,A,B,C} e P={S-> λ|aA|aB, S3->aA|aB, A->a|aS, B->aC, C->a|aS} = {S-> λ|aA|aB, A->a|aS, B->aC, C->a|aS}

Passiamo all’automa NDA che riconosce L(G): M = (Q, δ, q0, F) con alfabeto di ingresso X ove:

* Q = VU{q} = {S,A,B,C,q}
* q0=S
* F = {q, S}
* δ:QxX---->2^Q definita:
  + δ(S,a) = {A,B}
  + δ(A,a) = {S,q}
  + δ(B,a) = {C}
  + δ(C,a) = {S,q}

1. L’automa accettore FSA M’ equivalente ad M si costruisce come segue