1. Dati i linguaggi A e B, lo *shuffle perfetto* di A e B è il linguaggio

$$\{w \mid w = a_1 b_1 a_2 b_2 \dots a_k b_k, \text{ dove } a_1 a_2 \dots a_k \in A \text{ e } b_1 b_2 \dots b_k \in B, \text{ ogni } a_i, b_i \in \Sigma\}$$

Mostrare che la classe dei linguaggi regolari è chiusa rispetto allo shuffle perfetto, cioè che se A e B sono linguaggi regolari allora anche il loro shuffle perfetto è un linguaggio regolare.

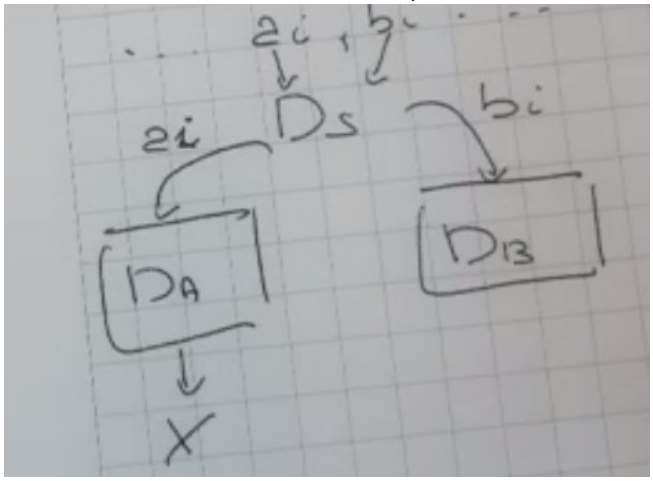
A,B sono regolari → Shuffle è regolare
a=|1|0|1|
b=|0|0|1|
w=|10|00|11

∑* = (0+1)* indica tutte le stringhe (compresa la vuota) ottenibili da linguaggio

Sappiamo quindi che esistono due automi del tipo:
$D_A = \{Q_A, \sum, \delta_A, q_A, F_A\}$
$D_B = \{Q_B, \sum, \delta_B, q_B, F_B\}$

Vogliamo produrre quindi:
$D_S = \{Q_S, \sum, \delta_S, q_S, F_S\}$

L'automa quindi sa l'ordine e che gli elementi sono invertiti, conseguentemente si va comunque almeno ad uno stato accettante.
L'idea informale è di costruire un automa che prende la posizione generica, richiama la procedura e poi non vedendo il funzionamento interno, richiamo i simboli avendo una dimostrazione generica:



Dim. formale

$D_S = \{Q_S, \sum, \delta_S, q_S, F_S\}$
Generica $\delta(q_X, a)$ → q

Ad esempio metto come input il simbolo "a":
$\delta(x,y,A), a) \rightarrow (\delta_A(X,a),y, B)$

L'idea quindi è che gli input si alternano e mi resta da aggiornare lo stato in cui si trova A, usando le sue funzioni di transizione.

x stato corrente D_A
y stato corrente D_B
A flag

$\delta((x, y, B), b) \rightarrow (x, \delta_B(y,b), A)$

$\delta(q,x) \rightarrow q$
L'idea è che la tupla rappresenti gli stati dell'automa, ottenendo gli stati nuovamente e riorganizzandoli, costruendo automaticamente la funzione di transizione di δ_S.

Quindi l'idea è di avere un aggiornamento degli stati tali da avere le transizioni ricombinate, rimanendo con gli input uguali

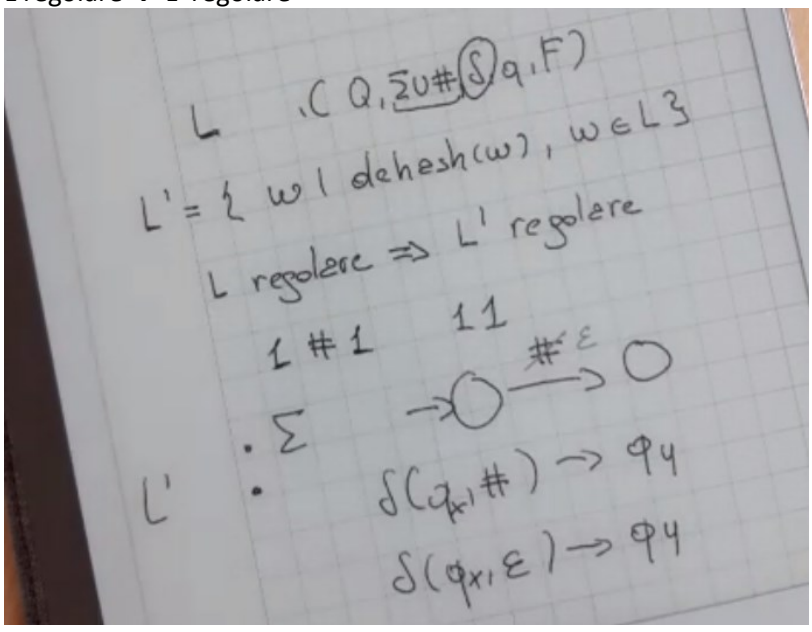
- Q_S
$Q_A \times Q_B \times \{A, B\}$
Usiamo quindi "x" come prodotto cartesiano, tipo avendo
$\{a,b\} \times \{c,d\}$ eseguo il prodotto tra insiemi (prodotto cartesiano)
$=\{(a,c), (a,d), (a,c), (b,d), (b,c)\}$

$\{q_x, q_y\} \times \{q_Z\} \times \{A, B\}$
$\{(q_X, q_Z), (q_Y, q_Z)\} \times \{A, B\}$
(q_X, q_Z, A)…..

La risposta è: creare gli stati come prodotto (producendo tutto "brutalmente" con tutte le possibili combinazioni).
Stato iniziale q_S
$q_S = (q_A,q_B,A)$
$\delta[(q_A,q_B,A), a_{1]} \rightarrow (\delta(q_A, q_1))$

$F_A \times F_B \times$
b_K
$\delta((q_X, q_Y, B), b_K)$

$L'=\{w \mid dehash(w), w \in L\}$
L regolare → L' regolare



L linguaggio regolare su $\sum$
$L' = \{y \mid xy \in L \text{ per } x \in \sum^*\}$
w=1011, $w \in L$

11
011
1011
ε

(Q, δ, q, F, ∑)

2. Sia A un linguaggio, e sia $DROPOUT(A)$ come il linguaggio contenente tutte le stringhe che possono essere ottenute togliendo un simbolo da una stringa di A:

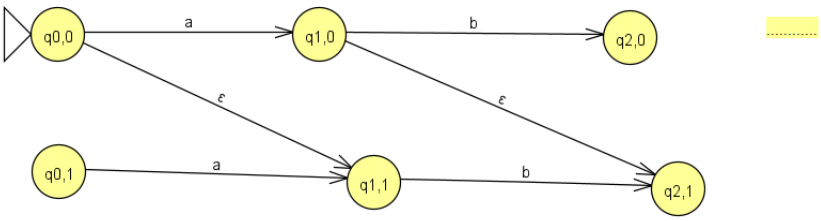
$$DROPOUT(A) = \{xz \mid xyz \in A \text{ dove } x, y \in \Sigma^* \text{ e } y \in \Sigma\}.$$

Mostrare che la classe dei linguaggi regolari è chiusa rispetto all'operazione $DROPOUT$, cioè che se A è un linguaggio regolare allora $DROPOUT(A)$ è un linguaggio regolare.

Quindi:
{xz | xyz ∈ A, x, z ∈ Z*, y ∈ Σ}
Sia M=(Q, Σ ,δ, q_0, F) un DFA che riconosce A.
M'=(Q', Σ , δ', q_0, F')
In pratica si saltano a random tutti gli elementi, andando in uno, un altro o un altro ancora, senza un ordine definito. Per ognuno deve ricordarsi che ha saltato quel carattere (non ne salta più di uno):
w_1w_2 .....w_r
q_0q_1.....q_n

Si introduce quindi un simbolo tale da capire se si è saltato o meno, eseguendo poi la transizione:
Q'=Q x {0,1} (q, 0) "non ho saltato"
q' = (q_0, 0) (q,1) "ho saltato"

Qui ragiono che per qualche a ∈ Σ
δ' ((q,0)a)={(δ(q_0,a),0)} (quindi ho un simbolo e vado avanti con 0)
δ((q,0)ε)={(p,1)|δ(q,a)=p (salto un simbolo oppure rimango sulla transizione stessa perché vuota)
δ' ((q,1)a)={(δ(q,a),1)} (quindi ho un simbolo e vado avanti con 1)



F'={(q1,1)|q_0 ∈ F}

3. Per una stringa $w = w_1w_2 \dots w_n$, l'*inversa* di w è la stringa $w^R = w_n \dots w_2w_1$. Per ogni linguaggio A, sia $A^R = \{w^R \mid w \in A\}$. Mostrare che se A è regolare allora lo è anche A^R.

Questo esercizio e' diverso da quelli visti in precedenza; dato un linguaggio regolare, ci viene chiesto di dimostare che con una modifica possiamo ottenere un altro linguaggio regolare. Dobbiamo percio' ragionare in modo piu' generico, senza produrre un automa specifico. Una soluzione informale e' la seguente

1. Dato il linguaggio A, produco un NFA_1 che lo riconosce A.
2. Per produrre l'automa che riconoscere il linguaggio A^R, posso modificare NFA_1 in NFA_2
   1. Inverto la direzione delle transizioni
   2. Lo stato iniziale di NFA^1 diventa accettante
   3. Lo stato accettante di NFA^2 diventa iniziale
3. Se NFA^1 ha piu' di uno stato accettante, aggiungo uno nuovo stato iniziale e una epsilon transizione pverso tutti gli ex-stati accettanti.

Per questo esercizio e' necessario partire da un NFA in modo da avere un solo percorso da stato iniziale ad accettante e poter seguire l'automa in senso inverso.

4. Sia $A/b = \{w \mid wb \in A\}$. Mostrare che se A è un linguaggio regolare e $b \in \Sigma$, allora A/b è regolare.

Per dimostrare che A/B è regolare, possiamo costruire un automa a stati finiti, in particolare un ε-NFA che, partendo dall'automa a stati finiti A=(Q U $\{q_0\}$', Σ, q_0, δ, F) che ha lo stesso insieme di stati, stesso stato iniziale e gli stessi stati finali. Essendo che si deve sempre essere la produzione di "w", lo stato iniziale conterrà, con l'aggiunta di una ε-transizione, un automa con tutti gli stati finali raggiungibili dal vecchio stato iniziale, con l'aggiunta della nuova stringa. Gli stati finali sono inoltre gli stessi.

5. Sia $A/B = \{w \mid wx \in A \text{ per qualche } x \in B\}$. Mostrare che se A è un linguaggio regolare e B un linguaggio qualsiasi, allora A/B è regolare.

Per dimostrare che A/B è regolare, possiamo costruire un automa a stati finiti, in particolare un ε-NFA che, partendo dall'automa a stati finiti A=(Q U $\{q_0\}$', Σ, q_0, δ, F) che ha lo stesso insieme di stati, stesso stato iniziale e gli stessi stati finali. Essendo che si deve sempre essere una stringa in più, lo stato iniziale conterrà, con l'aggiunta di una ε-transizione, un automa con tutti gli stati finali raggiungibili dal vecchio stato iniziale, con l'aggiunta della stringa prefissa. Gli stati finali sono inoltre gli stessi.

Il pumping lemma afferma che ogni linguaggio regolare ha una lunghezza del pumping p, tale che ogni stringa del linguaggio può essere iterata se ha lunghezza maggiore o uguale a p. La *lunghezza minima del pumping* per un linguaggio A è il più piccolo p che è una lunghezza del pumping per A. Per ognuno dei seguenti linguaggi, dare la lunghezza minima del pumping e giustificare la risposta.

(a) 110^*
(b) $1^*0^*1^*$
(c) $0^*1^*0^*1^* + 10^*1$
(d) $(01)^*$
(e) $\emptyset$
(f) $0^*01^*01^*$
(g) $10(11^*0)^*0$
(h) 101101
(i) $\{w \in \Sigma^* \mid w \neq 101101\}$
(j) 0001^*
(k) 0^*1^*
(l) $001 + 0^*1^*$
(m) ε
(n) $1^*01^*01^*$
(o) 1011
(p) Σ^*

(a) 3: si necessita di avere almeno tre caratteri per poter cominciare ad eseguire il pumping. Infatti parole come 00, 11 o similari non sono acettate.
(b) 3: parole come 11 (cioè xy^0z) non sono accettate per costruzione, prendendo ad esempio 101 come xyz si nota che devono esserci entrambi per poter cominciare a pompare.
(c) 3: L'unione c'è ma non ci interessa, la proprietà vale comunque. Inoltre si nota che devo avere almeno due 1 ma l'idea è la stessa dell'esercizio precedente.
(d) 1: l'insieme vuoto deve avere almeno una stringa con cui operare e quindi pompare all'infinito una parola.
(e) 2: abbiamo bisogno di entrambi i caratteri per pompare.
(f) 3: le parole devono essere divise e definite in 3 pezzi anche qui per formare una parola valida e pompare.
(g) 4: in pratica la lunghezza deve essere 4 in quanto potrebbe esserci una suddivisione che non sbilancia la stringa avendo soli tre caratteri, avendo stringhe che pompate non sarebbero nel linguaggio. Sapendo che (11*0)* è l'espressione minima, avremo quantomeno bisogno di 0 oppure 1 per cominciare a pompare.
(h) 4: si vede infatti che ipotizzando 3 come lunghezza minima, potremmo avere una stringa del tipo 111 che rimane sempre regolare.
(i) 3: se sappiamo (penso io) che la stringa precedente non fa parte del linguaggio, significa considerando l'alfabeto precedente che necessitiamo di almeno tre caratteri per poter avere una stringa pompabile, considerando il caso complementare a quello descritto sopra.
(j) 4: potremmo banalmente avere la stringa 000 che non sarebbe pompabile.
(k) 2: anche qui, scegliendo 1 non sarebbe pompabile; con 2 almeno avremmo il possibile caso 01, regolarmente pompabile.
(l) 3: lasciando stare il caso dell'unione, comunque si nota che la stringa 001 non può avere 2 come lunghezza minima pompabile.

(m) 1: la stringa vuota deve necessitare di almeno un carattere qualsiasi.
(n) 3: si considera infatti che la minima stringa effettivamente pompabile sia 001
(o) 5: di fatto 1011 potrebbe non essere accettata dal linguaggio come pompata, perché già integralmente parte del linguaggio stesso.
(p) 2: considerando che la stringa vuota necessita di almeno un carattere e l'alfabeto la comprende di sicuro essendo star, potrebbe bastare per un generico alfabeto avere un solo altro carattere.

7. Sia $\Sigma = \{0, 1\}$, e considerate il linguaggio

$$D = \{w \mid w \text{ contiene un ugual numero di occorrenze di 01 e di 10}\}$$

Mostrare che D è un linguaggio regolare.

L'obiettivo come al solito è di costruire un automa in grado di riconoscere questo linguaggio presentato.
L'alfabeto Σ = {0, 1} e lo stato iniziale. Partiamo quindi dall'automa A=(Q, Σ, q_0, δ, F) con una funzione di transizione costruita come:

- $\delta(x,y,01),a) = (\delta_1(x,a),y,01)$
- $\delta(x,y,10),a) = (\delta_1(x,a),y,10)$

perché devo avere ugual numero di occorrenze di 01/10.
L'insieme degli stati finali sarà costituito da:
$F = F_A \times F_B \times \{0,1\}$
Avendo come stato iniziale (partendo da A):
$q=(q_A,q_B,A)$
e avendo come stato accettante
$F = F_A \times F_B \times \{A\}$

8. Sia $\Sigma = \{0, 1\}$.

- Mostrare che il linguaggio $A = \{0^k u 0^k \mid k \geq 1 \text{ e } u \in \Sigma^*\}$ è regolare.
- Mostrare che il linguaggio $B = \{0^k 1 u 0^k \mid k \geq 1 \text{ e } u \in \Sigma^*\}$ non è regolare.

1) Assumiamo per assurdo il linguaggio sia regolare.
Ricordiamo che "u" è un simbolo della grammatica, quindi può essere sia 0 che 1
Allora esisterà una suddivisione w=xyz tale da avere x ≠ ε, |xy| <= k con
$w=0^p u 0^p$ avendo almeno xy completamente piena di 0.
Il pumping xy^iz non dimostra lo sbilanciamento, infatti semplicemente la stringa
rimane composta di egual numero di 0 e sempre la stringa u in mezzo.
Un qualsiasi pumping per k > 0, ad esempio.
$w=0^k u 0^{p-k}$ comunque non dimostra alcunché,
infatti abbiamo un numero di u sempre compreso tra un egual numero di zeri ed il linguaggio rimane sempre regolare.
2) Assumiamo per assurdo il linguaggio sia regolare.
Allora esisterà una suddivisione w=xyz tale da avere x ≠ ε, |xy| <= k con
u=0 e $w=0^p10^q0^k$
In questo caso, siccome $x=0^k$ ed $y=0^q$ allora avremo in "z" la rimanente parte della stringa, quindi:
$xy^0z = xz=0^k10^{k-p-q}$, quindi $0^{2k-p-q}1$
Si vede quindi che il numero di 0 è sbilanciato rispetto al numero di 1 e il linguaggio quindi non può essere regolare.

9. Dimostrare che i seguenti linguaggi non sono regolari.

(a) $\{0^n1^m0^m \mid m, n \geq 0\}$
(b) $\{0^n1^m \mid n \neq m\}$
(c) $\{w \in \{0,1\}^* \mid w \text{ non è palindroma}\}$
(d) $\{wtw \mid w, t \in \{0,1\}^+\}$, dove $\{0,1\}^+$ è l'insieme di tutte le stringhe binarie di lunghezza maggiore o uguale a 1

a) Assumiamo per assurdo il linguaggio sia regolare.
Allora esisterà una suddivisione w=xyz tale da avere x ≠ ε, |xy| <= k con
$w=0^k1^p0^k$
e con pumping i=0
avremo $0^{k-p}1^k0^k$ che chiaramente non appartiene al linguaggio e quindi non è regolare.

b) Assumiamo per assurdo il linguaggio sia regolare.
Allora esisterà una suddivisione w=xyz tale da avere x ≠ ε, |xy| <= k con
$w=0^p1^q$ con p > q o p < q, entrambi > 0 e < k
In questo caso è semplice perché basta letteralmente prendere x=ε, $y=0^p$ e $z=1^q$
tale che pompando i avremo sempre un numero > di 0 rispetto agli 1.
Banalmente la cosa è dimostrata anche nel caso xy^0z con n ≠ m e quindi
si dimostra che non è regolare.

c) Assumiamo per assurdo il linguaggio sia regolare.
Allora esisterà una suddivisione w=xyz tale da avere x ≠ ε, |xy| <= k con
$w=0^p1^q0^r$
In questo modo, eseguendo un pumping di i=0, avremo
$0^{p-r}1^q0^p$ che rivela la parola non sia nel linguaggio e dunque non è regolare.

d) Assumiamo per assurdo il linguaggio sia regolare.
Allora esisterà una suddivisione w=xyz tale da avere x ≠ ε, |xy| <= k con
$0^k1^p0^k$. Eseguendo il pumping, ad esempio con i=2
e scegliendo $y=0^k1^p$ avremmo una cosa del tipo $0^{2k+2p}0^k$ che non rispetta |xy| <= k
nel numero di 0 ed il linguaggio non è regolare.

10. Per ogni linguaggio A, sia $SUFFIX(A) = \{v \mid uv \in A \text{ per qualche stringa } u\}$. Mostrare che la classe dei linguaggi context-free è chiusa rispetto all'operazione di $SUFFIX$.

Sappiamo che L è context-free. Esiste quindi una grammatica G in forma normale del tipo::
A → BC
D → d
Se A → BC allora possiamo scrivere una parola w=bc
Il suffisso che è parte della parola, potrà essere:

- caso 1 (prefisso nullo) BC
- caso 2 (prefisso che è una parte di B): B'C
- caso 3 (prefisso che è esattamente B): C
- caso 4 (prefisso che è B più una parte di C): C'
- caso 5 (ho solo prefisso): ε

Ci sarà quindi la regola:
A → BC |B'C|C|C'|ε
D' → d| ε

La grammatica G' quindi avrà S' come stato iniziale e condividerà gli stessi stati per le stesse regole.
Così descritta la grammatica G' è context-free in quanto in CNF.

11. Usa i linguaggi $A = \{a^m b^n c^n \mid m, n \geq 0\}$ e $B = \{a^n b^n c^m \mid m, n \geq 0\}$ per mostrare che la classe dei linguaggi context-free non è chiusa per intersezione.

L'operazione di intersezione è definita liberamente per i linguaggi regolari; significa quindi che se eseguiamo l'intersezione di due linguaggi context-free, in questo caso A e B, anch'essa deve essere CF.
Considerando ad esempio due generiche parole del linguaggio $w=a^pb^qc^p$ e $w=a^qb^pc^q$

- caso base, con p=q=0 sono: abc & abc, dunque l'intersezione è fattibile
- caso induttivo, con generici esponenti p, q entrambi > 0 possiamo avere:
  (es. p=2, q=3)
  $a^2b^3c^2$ & $a^3b^2c^3$
  avremo che l'intersezione genererà numero diverso di a, di b, e di c.
  Quindi ad esempio potremmo avere stringhe del tipo:
  $a^2b^5c^3$, $a^5b^3c^3$, ecc.

Eseguendo l'intersezione si nota che otterremmo stringhe diverse e, se volessimo dimostrarlo con il PL per linguaggi CF, otterremmo sempre stringhe sbilanciate. Dunque l'operazione di intersezione non può essere chiusa per il linguaggi CF.

12. Dimostrare che i seguenti linguaggi sono context-free. Salvo quando specificato diversamente, l'alfabeto è $\Sigma = \{0, 1\}$.

(a) $\{w \mid w$ contiene almeno tre simboli uguali a $1\}$
(b) $\{w \mid$ la lunghezza di w è dispari$\}$
(c) $\{w \mid w$ inizia e termina con lo stesso simbolo $\}$
(d) $\{w \mid$ la lunghezza di w è dispari e il suo simbolo centrale è $0\}$
(e) $\{w \mid w = w^R$, cioè w è palindroma$\}$
(f) $\{w \mid w$ contiene un numero maggiore di 0 che di $1\}$
(g) Il complemento di $\{0^n 1^n \mid n \geq 0\}$
(h) Sull'alfabeto $\Sigma = \{0, 1, \#\}$, $\{w\#x \mid w^R$ è una sottostringa di x e $w, x \in \{0, 1\}^*\}$
(i) $\{x\#y \mid x, y \in \{0, 1\}^*$ e $x \neq y\}$
(j) $\{xy \mid x, y \in \{0, 1\}^*$ e $|x| = |y|$ ma $x \neq y\}$
(k) $\{a^i b^j \mid i \neq j$ e $2i \neq j\}$

Per dimostrare che i linguaggi *sono* context-free dobbiamo necessariamente fare delle derivazioni.
Se dovessimo dimostrare che *non* sono context-free allora dovremmo usare il PL per CFL.

a) S → 1X1X1X1X
   X → 0X|1X|ε
b) S → 0A|1A
   A → 0S|1S|ε
c) S → 0A|1B
   A → 0|ε
   B → 1|ε
d) S → 0S0|1S1|0|0S1|1S0
e) S → 0|1|1S1|0S0|ε
f) S → T0T
   T → 1T0|0T1|0T0|ε
g) S → 1A|0A
   A → A0|
h) S → A
   A → #B|0A0|1A1
   B → 0B|1B|ε
i) S → A#B|B#A|ε
   A → TAT|0
   B → TBT|1
   T → 0|1
j) S → AB|BA

A → 0|0A0|0A1|1A0|1A1
B → 1|0B0|0B1|1B0|1B1

k) S → S_1|S_2
S_1 → aA
A → bAb|aA|ε
S_2 → Bb!aBb
B → Bb|aaBb|aBb|ε

13. Se A e B sono linguaggi, definiamo $A \circ B = \{xy \mid x \in A, y \in B \text{ e } |x| = |y|\}$. Mostrare che se A e B sono linguaggi regolari, allora $A \circ B$ è un linguaggio context-free.

Se A e B sono linguaggi regolari, allora sono descrivibili mediante le medesime operazioni dei linguaggi regolari e successivamente almeno da un automa DFA/NFA.
Sia M=(Q, Σ ,δ, q_o, F) un DFA che riconosce A.
M'=(Q', Σ , δ', q_o, F')
Descriviamo quindi A, composto evidentemente da stringhe "x", poi B, composto da stringhe di tipo "y"
Significa quindi che per questi dettagliamo:
(r_A, L) → (s_A,M) se $\delta_L(r_A, x)=s_A$
(r_B, L) → (s_B,M) se $\delta_L(r_B, y)=s_A$
Similmente è definita (r_A, L, M) → (s_b,M, L) se $\delta_L(r_A, x)=\delta_L(r_B, y)$.
A queste condizioni notiamo che il linguaggio context-free necessariamente richiede che gli stati finali corrispondano tra i due linguaggi.

Articoliamo quindi come idea di stato finale F (δ_0, x), F(δ_1, y) = F(X • Y)

Necessariamente l'idea è che, nel caso non esista uno dei due stati,
lo stato finale deve essere:

F(X • Y) = F (δ_0, x)

oppure

F(X • Y) = F(δ_1, y)

per la caratterizzazione di termini di cui sopra. Dunque il linguaggio è CF.

14. Dimostrare che se G è una CFG in forma normale di Chomsky, allora per ogni stringa $w \in L(G)$ di lunghezza $n \geq 1$, ogni derivazione di w richiede esattamente $2n - 1$ passi.

Sapendo che G è in forma normale di Chomsky, quindi nella forma
A→ BC
A → a

Prendiamo l'esempio di una grammatica di un es. fatto in questo file:
S' → AX|YB|a|AS|SA
S → AX|YB|a|AS|SA
A → b|AX|YB|a|AS|SA
B→ b
X → SA
Y → a

Consideriamo una stringa di lunghezza 1, possibilmente composta da una situazione del tipo
A → BC
Sono tutti simboli terminali e quindi, ciascuno è in forma normale di Chomsky, non avendo regole unitarie/transizioni vuote/più di tre transizioni per regola.
Induttivamente, nel caso di n > 1, avremo una situazione in cui, per poter applicare Chomsky abbiamo bisogno di almeno 2 regole, una terminale e non terminale ed eventuali altre regole.

Quindi:
A → BC
B → b
C → b|c|d
Ogni singola derivazione, per Chomsky, richiede esattamente l'applicazione di una sostituzione in altra regola e ogni regola può essere composta fino ad un massimo di 2 regole.
Essendoci almeno una regola terminale, per ipotesi, il numero di stringhe necessarie a comporre tale situazione sarà 2n.
Data appunto la presenza di almeno una regola terminale, il numero di stringhe necessarie a mostrare tale situazione in una derivazione si compone di $2n - 1$ passi.

In una situazione reale come la grammatica ottenuta sopra, si nota che per ogni simbolo di partenza ne corrispondono almeno 2 derivati, con l'aggiunta della regola terminale.
Consideriamo quindi S' → AB ed S → XY. Tale situazione comporta due regole non terminali ed un'altra possibilmente terminale. Dunque generalmente Chomsky richiede questo tipo di derivazione.