3. Considera una generalizzazione delle grammatiche context-free che consente di avere espressioni regolari sul lato destro delle regole di produzione. Senza perdita di generalità, puoi assumere che per ogni variabile $A \in V$, la grammatica generalizzata contenga un'unica espressione regolare $R(A)$ su $V \cup \Sigma$. Per applicare una regola di produzione, scegliamo una variable A e la sostituiamo con una parola del linguaggio descritto da $R(A)$. Come al solito, il linguaggio della grammatica generalizzata è l'insieme di tutte le stringhe che possono essere derivate dalla variabile iniziale.

Per esempio, la seguente grammatica generalizzata descrive il linguaggio di tutte le espressioni regolari sull'alfabeto $\{0, 1\}$. I simboli in rosso sono terminali, i simboli in nero sono variabili oppure operatori.

$S \to (T+)^*T + \emptyset$	(Espressioni regolari)
$T \to \varepsilon + F^*F$	(Termini = espressioni che si possono sommare)
$F \to (0 + 1 + (S))(* + \varepsilon)$	(Fattori = espressioni che si possono concatenare)

Dimostra che ogni grammatica context-free generalizzata descrive un linguaggio context-free. In altre parole, dimostra che consentire espressioni regolari nelle regole di produzione non aumenta il potere espressivo delle grammatiche context-free.

3. Per dimostrare che ogni grammatica context-free generalizzata descrive un linguaggio context-free dobbiamo dimostrare che le grammatiche generalizzate sono equivalenti alle normali grammatiche context free.

È facile vedere che le grammatiche context-free sono un caso particolare di grammatiche context-free generalizzate: una regola $A \to u$ dove u è una stringa di variabili e terminali è una regola valida anche per le grammatiche generalizzate.

Dimostreremo che consentire espressioni regolari nelle regole non aumenta il potere espressivo delle grammatiche mostrando come possiamo costruire una grammatica context-free equivalente ad una grammatica generalizzata. La costruzione procede rimpiazzando le regole con espressioni regolari con altre regole equivalenti, finché tutte le regole sono nella forma semplice consentita nelle grammatiche context-free normali:

(a) rimpiazza ogni regola $A \to R + S$ con le due regole $A \to R$ e $A \to S$;

(b) per ogni regola $A \to R.S$, aggiungi due nuove variabili A_R e A_S e rimpiazza la regola con le regole $A \to A_R A_S$, $A \to R$ e $A \to S$;

(c) per ogni regola $A \to S^*$, aggiungi una nuova variabile A_S e rimpiazza la regola con le regole $A \to A_S A$, $A \to \varepsilon$ e $A_S \to S$;

(d) rimpiazza ogni regola $A \to \emptyset$ con la regola $A \to A$;

(e) ripeti da (a) finché non rimangono solamente regole del tipo $A \to u$ dove u è una stringa di variabili e terminali, oppure $A \to \varepsilon$.

Ogni passaggio della costruzione modifica la grammatica in modo da essere sicuri che generi lo stesso linguaggio. Inoltre, la costruzione termina quando tutte le regole sono nella forma "standard" $A \to u$, senza operatori regolari.

3. *Dimostra che se L è un linguaggio context-free, allora anche L^R è un linguaggio context-free.*

Se L è un linguaggio context free allora esiste una grammatica G che lo genera. Possiamo assumere che G sia in forma normale di Chomksy. Di conseguenza le regole di G sono solamente di due tipi: $A \to BC$, con A, B, C simboli non terminali, oppure $A \to b$ con b simbolo nonterminale.

Costruiamo la grammatica G^R che genera L^R in questo modo:

- ogni regola $A \to BC$ viene sostituita dalla regola $A \to CB$;
- le regole $A \to b$ rimangono invariate.

2. **(8 punti)** *Per ogni linguaggio L, sia* $\text{prefix}(L) = \{u \mid uv \in L \text{ per qualche stringa } v\}$. *Dimostra che se L è un linguaggio context-free, allora anche* $\text{prefix}(L)$ *è un linguaggio context-free.*

   Se L è un linguaggio context-free, allora esiste una grammatica G *in forma normale di Chomski* che lo genera. Possiamo costruire una grammatica G' che genera il linguaggio $\text{prefix}(L)$ in questo modo:

   - per ogni variabile V di G, G' contiene sia la variabile V che una nuova variabile V'. La variabile V' viene usata per generare i prefissi delle parole che sono generate da V;
   - tutte le regole di G sono anche regole di G';
   - per ogni variabile V di G, le regole $V' \to V$ e $V' \to \varepsilon$ appartengono a G;
   - per ogni regola $V \to AB$ di G, le regole $V' \to AB'$ e $V' \to A'$ appartengono a G';
   - se S è la variabile iniziale di G, allora S' è la variabile iniziale di G'.

3. Per ogni linguaggio L, sia $\text{suffix}(L) = \{v \mid uv \in L \text{ per qualche stringa } u\}$. Dimostra che se L è un linguaggio context-free, allora anche $\text{suffix}(L)$ è un linguaggio context-free.

Per dimostrare che suffix(L) è un linguaggio context-free se L è context-free:

Sia G = (V, Σ, R, S) la grammatica context-free che genera L. Costruiamo una nuova grammatica G' = (V', Σ, R', S') che genera suffix(L):

V' = V ∪ {S'} dove S' è un nuovo simbolo non terminale R' contiene tutte le regole di R, più le seguenti nuove regole:

- S' → S
- S' → aS' per ogni a ∈ Σ

Questa grammatica G' genera suffix(L) perché:

- Può generare qualsiasi stringa in L (usando S' → S e poi le regole originali)
- Può aggiungere qualsiasi prefisso arbitrario a una stringa di L (usando ripetutamente S' → aS')

Poiché G' è una grammatica context-free, suffix(L) è un linguaggio context-free.

2. **(8 punti)** Per ogni linguaggio L sull'alfabeto Σ, sia $\text{superstring}(L) = \{xyz \mid y \in L \text{ e } x, z \in \Sigma^*\}$. Dimostra che se L è un linguaggio context-free, allora anche $\text{superstring}(L)$ è un linguaggio context-free.

Sia G = (V, Σ, R, S) la grammatica context-free che genera L. Costruiamo una nuova grammatica G' = (V', Σ, R', S') che genera superstring(L):

V' = V ∪ {S', A, B} dove S', A, B sono nuovi simboli non terminali R' contiene tutte le regole di R, più le seguenti nuove regole:

- S' → ASB
- A → aA | ε per ogni a ∈ Σ
- B → bB | ε per ogni b ∈ Σ

Questa grammatica G' genera superstring(L) perché:

- Può generare qualsiasi prefisso x ∈ Σ* usando le regole di A
- Può generare qualsiasi stringa y ∈ L usando le regole originali di G
- Può generare qualsiasi suffisso z ∈ Σ* usando le regole di B

Poiché G' è una grammatica context-free, superstring(L) è un linguaggio context-free.

2. (8 punti) Per ogni linguaggio L, sia substring$(L) = \{v \mid uvw \in L$ per qualche coppia di stringhe $u, w\}$. Dimostra che se L è un linguaggio context-free, allora anche substring(L) è un linguaggio context-free.

Sia $G = (V, \Sigma, R, S)$ la grammatica context-free che genera L. Costruiamo una nuova grammatica $G' = (V', \Sigma, R', S')$ che genera $\mathbf{substring}(L)$:

- $V' = V \cup \{S', A, B\}$, dove S', A, e B sono nuovi simboli non terminali.
- R' contiene tutte le regole di R, più le seguenti nuove regole:
  - $S' \to ASB$
  - $A \to aA \mid \epsilon$ per ogni $a \in \Sigma$
  - $B \to bB \mid \epsilon$ per ogni $b \in \Sigma$

Questa grammatica G' genera $\mathbf{substring}(L)$ perché:

- Può generare qualsiasi prefisso $u \in \Sigma^*$ usando le regole di A.
- Può generare qualsiasi stringa $v \in L$ usando le regole originali di G.
- Può generare qualsiasi suffisso $w \in \Sigma^*$ usando le regole di B.

Poiché G' è una grammatica context-free, $\mathbf{substring}(L)$ è un linguaggio context-free.

3. (12 punti) Dimostra che se $L \subseteq \Sigma^*$ è un linguaggio context-free allora anche il linguaggio

$$censor(L) = \{\#^{|w|} \mid w \in L\}$$

è un linguaggio context-free.

Soluzione: Se L è un linguaggio context-free, allora esiste una grammatica $G = (V, \Sigma, R, S)$ che lo genera. Possiamo assumere che questa grammatica sia in forma normale di Chomsky. Per dimostrare che $censor(L)$ è context-free, dobbiamo essere in grado di definire una grammatica che possa generarlo. Questa grammatica è una quadrupla $G' = (V', \Sigma', R', S')$ definita come segue.

- L'alfabeto contiene solo #: $\Sigma' = \{\#\}$.
- L'insieme di variabili è lo stesso della grammatica G: $V' = V$.
- Il nuovo insieme di regole R' è ottenuto rimpiazzando ogni regola nella forma $A \to b$, con b simbolo terminale, con la regola $A \to \#$, e lasciando invariate le regole nella forma $A \to BC$ e la regola $S \to \varepsilon$ (se presente).
- La variabile iniziale rimane la stessa: $S' = S$.

Data una derivazione $S \Rightarrow^* w$ della grammatica G possiamo costruire una derivazione nella nuova grammatica G' che applica le stesse regole nello stesso ordine, e che deriva una parola dove ogni simbolo terminale di w è rimpiazzato da #. Quindi, G' permette di derivare tutte le parole in $censor(L)$. Viceversa, data una derivazione $S \Rightarrow^* \#^n$ della nuova grammatica G' possiamo costruire una derivazione nella grammatica G che applica le stesse regole nello stesso ordine, e che deriva una parola dove ogni # è rimpiazzato da qualche simbolo terminale in Σ. Quindi, G' permette di derivare solo parole che appartengono a $censor(L)$.

3. (12 punti) Dimostra che se $L \subseteq \Sigma^*$ è un linguaggio context-free allora anche L^R è un linguaggio context-free, dove $L^R = \{w^R \in \Sigma^* \mid w \in L \text{ e } w^R \text{ è la stringa } w \text{ rovesciata}\}$.

Soluzione: Se L è un linguaggio context-free, allora esiste una grammatica $G = (V, \Sigma, R, S)$ che lo genera. Per dimostrare che L^R è context-free, dobbiamo essere in grado di definire una grammatica che possa generarlo. Questa grammatica è una quadrupla $G' = (V', \Sigma', R', S')$ definita come segue.

- L'alfabeto è lo stesso del linguaggio L originale: $\Sigma' = \Sigma$.
- L'insieme di variabili è lo stesso della grammatica G: $V' = V$.
- Il nuovo insieme di regole R' è ottenuto "rovesciando" la parte destra delle regole di R: $R' = \{A \to u^R \mid A \to u \in R\}$. In questo modo qualsiasi derivazione deve ora seguire le regole rovesciate.
- Si noti che mentre la parte destra delle regole deve rovesciata, l'ordine delle regole nella derivazione non deve essere invertito. Pertanto, la variabile iniziale rimane la stessa: $S' = S$.

3. (12 punti) Dimostra che se $L \subseteq \Sigma^*$ è un linguaggio context-free allora anche il seguente linguaggio è context-free:

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

dove $dehash(w)$ è la stringa che si ottiene cancellando ogni # da w.

Soluzione: Se L è un linguaggio context-free, allora esiste una grammatica $G = (V, \Sigma, R, S)$ che lo genera. Possiamo assumere che questa grammatica sia in forma normale di Chomsky. Per dimostrare che $dehash(L)$ è context-free, dobbiamo essere in grado di definire una grammatica che possa generarlo. Questa grammatica è una quadrupla $G' = (V', \Sigma', R', S')$ definita come segue.

- L'alfabeto tutti i simboli di Σ tranne #: $\Sigma' = \Sigma \setminus \{\#\}$.
- L'insieme di variabili è lo stesso della grammatica G: $V' = V$.
- Il nuovo insieme di regole R' è ottenuto rimpiazzando ogni regola nella forma $A \to \#$ con la regola $A \to \varepsilon$, e lasciando invariate le regole nella forma $A \to BC$, le regole nella forma $A \to b$ quando $b \neq \#$, e la regola $S \to \varepsilon$ (se presente).
- La variabile iniziale rimane la stessa: $S' = S$.

Data una derivazione $S \Rightarrow^* w$ della grammatica G possiamo costruire una derivazione nella nuova grammatica G' che applica le stesse regole nello stesso ordine, e che deriva una parola dove ogni # è rimpiazzato dalla parola vuota ε. Quindi, G' permette di derivare tutte le parole in $dehash(L)$.

Viceversa, data una derivazione $S \Rightarrow^* w$ della nuova grammatica G' possiamo costruire una derivazione nella grammatica G che applica le stesse regole nello stesso ordine. Di conseguenza, in ogni punto in cui la derivazione per G' applica la regola modificata $A \to \varepsilon$, la derivazione per G applicherà la regola $A \to \#$ inserendo un # in qualche punto della parola w. Al termine della derivazione si ottiene una parola w' tale che $dehash(w') = w$. Quindi, G' permette di derivare solo parole che appartengono a $dehash(L)$.

2. (9 punti) Considera la seguente funzione da $\{0,1\}^*$ a $\{0,1\}^*$:

$$\text{stutter}(w) = \begin{cases} \varepsilon & \text{se } w = \varepsilon \\ aa.\text{stutter}(x) & \text{se } w = ax \text{ per qualche simbolo } a \text{ e parola } x \end{cases}$$

Dimostra che se L è un linguaggio context-free sull'alfabeto $\{0,1\}$, allora anche il seguente linguaggio è context-free:

$$\text{stutter}(L) = \{\text{stutter}(w) \mid w \in L\}.$$

Per dimostrare che stutter(L) è context-free se L è context-free, costruiamo una grammatica context-free per stutter(L) a partire da una grammatica G per L.

Sia G = (V, Σ, R, S) la grammatica per L. Costruiamo G' = (V', Σ, R', S') per stutter(L):

V' = V ∪ {S', A_a | a ∈ Σ} S' è il nuovo simbolo iniziale R' contiene le seguenti regole:

- S' → S
- S' → A_a S per ogni a ∈ Σ
- A_a → aa per ogni a ∈ Σ
- Tutte le regole di R

G' genera stutter(L) perché:

- Può generare qualsiasi stringa di L (usando S' → S e le regole di R)
- Può inserire coppie di simboli identici in qualsiasi posizione (usando A_a → aa)

Quindi stutter(L) è context-free.

2. (9 punti) Dimostra che se L è un linguaggio context-free, allora anche il seguente linguaggio è context-free:

$$\text{delete}_{\#}(L) = \{xy \mid x\#y \in L\}.$$

Per dimostrare che delete#(L) = {xy | x#y ∈ L} è context-free se L è context-free:

Sia G = (V, Σ ∪ {#}, R, S) la grammatica per L. Costruiamo G' = (V', Σ, R', S') per delete#(L):

V' = V ∪ {S'} S' è il nuovo simbolo iniziale R' contiene le seguenti regole:

- S' → S
- Per ogni regola A → αBβ in R, dove α e β non contengono #, aggiungiamo A → αβ in R'
- Per ogni regola A → α#β in R, aggiungiamo A → αβ in R'

G' genera delete#(L) perché simula G ma "salta" il simbolo # quando lo incontra.

2. (9 punti) Dimostra che se L è un linguaggio context-free, allora anche il seguente linguaggio è context-free:

$$\text{insert}_{\#}(L) = \{x\#y \mid xy \in L\}.$$

Per dimostrare che insert#(L) = {x#y | xy ∈ L} è context-free se L è context-free:

Sia G = (V, Σ, R, S) la grammatica per L. Costruiamo G' = (V', Σ ∪ {#}, R', S') per insert#(L):

V' = V ∪ {S'} ∪ {A_# | A ∈ V} S' è il nuovo simbolo iniziale R' contiene le seguenti regole:

- S' → S_#
- Per ogni A → α in R, aggiungiamo A_# → α_#
- Per ogni simbolo terminale a ∈ Σ, aggiungiamo a_# → a#a e a_# → a
- Per ogni variabile A ∈ V, aggiungiamo A_# → A#

G' genera insert#(L) perché simula G ma permette di inserire # in qualsiasi posizione.

> **3. (12 punti)** Date due stringhe w e t, diciamo che t è una *permutazione* di w se t e ha gli stessi simboli di w con ugual numero di occorrenze, ma eventualmente in un ordine diverso. Per esempio, le stringhe 01011,e 00111 sono entrambe permutazioni di 11001.
>
> Dimostra che se $B \subseteq \{0,1\}^*$ è un linguaggio regolare, allora il linguaggio
>
> $$\text{SCRAMBLE}(B) = \{t \in \{0,1\}^* \mid t \text{ è una permutazione di qualche } w \in B\}$$
>
> è un linguaggio context-free.

Dimostrazione: Sia A = (Q, {0,1}, δ, q0, F) un DFA che riconosce B. Costruiamo una grammatica context-free G = (V, {0,1}, R, S) per SCRAMBLE(B) come segue:

V = {S} ∪ {[p,q,n0,n1] | p,q ∈ Q, n0,n1 ≥ 0}

Le regole R sono:

1. S → [q0,qf,0,0] per ogni qf ∈ F
2. [p,q,n0,n1] → 0[p,q,n0+1,n1] | 1[p,q,n0,n1+1] per ogni p,q ∈ Q
3. [p,q,n0+1,n1] → 0[δ(p,0),q,n0,n1] per ogni p,q ∈ Q
4. [p,q,n0,n1+1] → 1[δ(p,1),q,n0,n1] per ogni p,q ∈ Q
5. [p,p,0,0] → ε per ogni p ∈ Q

L'idea è che [p,q,n0,n1] rappresenta uno stato in cui:

- p è lo stato corrente in A
- q è lo stato finale che vogliamo raggiungere
- n0 e n1 sono i numeri di 0 e 1 ancora da leggere

Le regole 2-4 permettono di generare e consumare 0 e 1 in qualsiasi ordine, simulando una permutazione. La regola 5 permette di terminare quando abbiamo raggiunto lo stato desiderato e consumato tutti i simboli.

Per dimostrare la correttezza:

- Se w ∈ B, allora esiste una derivazione in G che genera ogni permutazione di w.
- Se t può essere derivato in G, allora esiste w ∈ B tale che t è una permutazione di w.

Quindi, SCRAMBLE(B) è generato da una grammatica context-free, e quindi è un linguaggio context-free.

3. (12 punti) Dati due linguaggi A, B, definiamo il linguaggio $MIX(A, B)$ come

$$MIX(A, B) = \{x_1 y_1 x_2 y_2 \dots x_n y_n \mid n \geq 0, x_i \in A, y_i \in B\}.$$

Si noti che ciascun x_i, y_i è una *stringa.* is a string. Dimostra che la classe dei linguaggi context free è chiusa per l'operazione MIX.

Dimostrazione: Siano A e B linguaggi context-free. Esistono quindi grammatiche context-free G_A = (V_A, Σ_A, R_A, S_A) e G_B = (V_B, Σ_B, R_B, S_B) che generano A e B rispettivamente.

Costruiamo una grammatica context-free G = (V, Σ, R, S) per MIX(A,B):

V = V_A ∪ V_B ∪ {S, X, Y} dove S, X, Y sono nuovi simboli non in V_A o V_B Σ = Σ_A ∪ Σ_B S è il nuovo simbolo iniziale R contiene tutte le regole in R_A e R_B, più le seguenti nuove regole:

1. S → XY | ε
2. X → S_AX | ε
3. Y → S_BY | ε

Per dimostrare la correttezza di G:

1. Se w ∈ MIX(A,B), allora w = x_1y_1...x_ny_n con x_i ∈ A, y_i ∈ B. Possiamo derivare w in G usando le regole 1-3 per generare la struttura alternata, e poi le regole di G_A e G_B per generare le stringhe x_i e y_i rispettivamente.
2. Se w può essere derivato in G, allora per la struttura delle regole, w avrà la forma x_1y_1...x_ny_n dove ogni x_i è derivabile in G_A e ogni y_i è derivabile in G_B. Quindi w ∈ MIX(A,B).

Poiché G è una grammatica context-free che genera MIX(A,B), MIX(A,B) è un linguaggio context-free.

Quindi, la classe dei linguaggi context-free è chiusa per l'operazione MIX.

3. (12 punti) Sia σ un alfabeto finito. Data stringa $w = w_1 \dots w_n$, dove ogni $w_i \in \Sigma$, definiamo $Rep(w) = w_1 w_1 w_2 w_2 \dots w_n w_n$. Cioè, $Rep(w)$ è la stringa ottenuta ripetendo ogni carattere di w. Dato un linguaggio $L \subseteq \Sigma^*$, definiamo il linguaggio

$$Rep(L) = \{Rep(w) \mid w \in L\}.$$

Dimostra che la classe dei linguaggi context free è chiusa per l'operazione Rep.

Dimostrazione: Sia L un linguaggio context-free. Esiste quindi una grammatica context-free G = (V, Σ, R, S) che genera L.

Costruiamo una grammatica context-free G' = (V', Σ, R', S') per Rep(L):

V' = V ∪ {S', A} dove S' e A sono nuovi simboli non in V S' è il nuovo simbolo iniziale R' contiene tutte le regole in R, più le seguenti nuove regole:

1. S' → SA
2. A → SA | ε
3. Per ogni a ∈ Σ, aggiungiamo la regola a → aa

Per dimostrare la correttezza di G':

1. Se w ∈ Rep(L), allora esiste una stringa v ∈ L tale che w = Rep(v). Possiamo derivare w in G' usando le regole 1-2 per generare la struttura di v, e poi la regola 3 per raddoppiare ogni carattere.
2. Se w può essere derivato in G', allora per la struttura delle regole, w avrà la forma Rep(v) dove v è derivabile in G. Quindi w ∈ Rep(L).

Poiché G' è una grammatica context-free che genera Rep(L), Rep(L) è un linguaggio context-free.

Quindi, la classe dei linguaggi context-free è chiusa per l'operazione Rep.

3. (12 punti) Dimostra che se B è un linguaggio regolare, allora il linguaggio

$$\text{PALINDROMIZE}(B) = \{ww^R \mid w \in B\}$$

è un linguaggio context-free.

Dimostrazione: Poiché B è regolare, esiste un DFA A = (Q, Σ, δ, q0, F) che riconosce B. Costruiamo una grammatica context-free G = (V, Σ, R, S) per PALINDROMIZE(B):

V = {S} ∪ {[q, r] | q, r ∈ Q} S è il simbolo iniziale R contiene le seguenti regole:

1. S → [q0, q] per ogni q ∈ F
2. [q, r] → a[δ(q, a), r]a per ogni q, r ∈ Q e a ∈ Σ
3. [q, q] → ε per ogni q ∈ Q

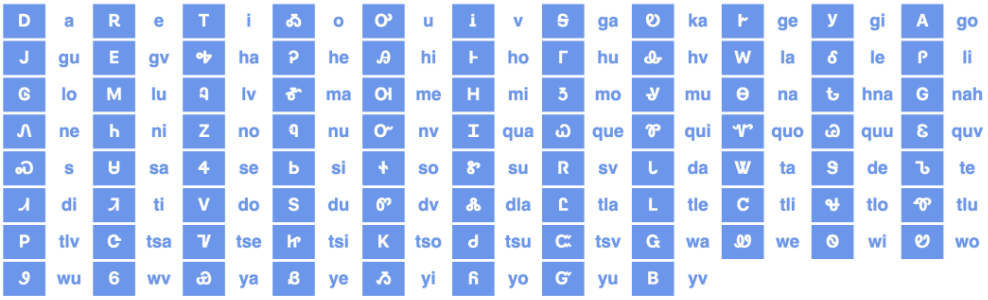
L'idea è che [q, r] genera tutte le stringhe w tali che δ*(q, w) = r, dove δ* è l'estensione di δ alle stringhe.

Per dimostrare la correttezza di G:

1. Se ww^R ∈ PALINDROMIZE(B), allora w ∈ B. Quindi, esiste una sequenza di stati q0, q1, ..., qn = qf in A tale che δ(qi, wi) = qi+1 e qf ∈ F. Possiamo derivare ww^R in G usando le regole 1, 2 ripetutamente, e infine 3.
2. Se una stringa può essere derivata in G, per la struttura delle regole, deve essere della forma ww^R dove w porta A dallo stato iniziale a uno stato finale, quindi w ∈ B e ww^R ∈ PALINDROMIZE(B).

Poiché G è una grammatica context-free che genera PALINDROMIZE(B), PALINDROMIZE(B) è un linguaggio context-free.

2. (9 punti) La *traslitterazione* è un tipo di conversione di un testo da una scrittura a un'altra che prevede la sostituzione di lettere secondo modalità prevedibili. La tabella seguente mostra il sistema di traslitterazione che permette di convertire la scrittura Cherokee nell'alfabeto latino:



Dati due alfabeti Σ e Γ, possiamo definire formalmente una traslitterazione come una funzione $T : \Sigma \mapsto \Gamma^*$ che mappa ogni simbolo di Σ in una stringa di simboli in Γ.

Dimostra che se $L \subseteq \Sigma^*$ è un linguaggio *context-free* e T è una traslitterazione, allora anche il seguente linguaggio è *context-free*:

$$T(L) = \{w \in \Gamma^* \mid w = T(a_0)T(a_1)\dots T(a_n) \text{ per qualche } a_0a_1\dots a_n \in L\}.$$

Sia L un linguaggio context-free su Σ.

Esiste quindi una grammatica context-free G tale che L = L(G).

Costruiamo una nuova grammatica G' su Γ come segue:

- G' ha gli stessi non-terminali di G

- Per ogni produzione A → α in G, G' ha la produzione A → T(α),

  dove T è applicata a ogni simbolo di α

- Gli stessi simboli iniziali

G' genera esattamente le stringhe T(w) per ogni w ∈ L(G).

Infatti, se S =>* w in G, allora S =>* T(w) in G',

e viceversa se S =>* w' in G', w' = T(w) per qualche w tale che S =>* w in G.

Quindi L(G') = T(L(G)) = T(L).

Poiché G' è context-free, anche T(L) lo è.