3. (12 punti) Una grammatica context-free è *lineare* se ogni regola in R è nella forma $A \to aBc$ o $A \to a$ per qualche $a, c \in \Sigma \cup \{\varepsilon\}$ e $A, B \in V$. I linguaggi generati dalle grammatiche lineari sono detti *linguaggi lineari*. Dimostra che i linguaggi regolari sono un sottoinsieme proprio dei linguaggi lineari.

Soluzione. Per risolvere l'esercizio dobbiamo dimostrare che ogni linguaggio regolare è anche un linguaggio lineare (i linguaggi regolari sono un sottoinsieme dei linguaggi lineari), e che esistono linguaggi lineari che non sono regolari (l'inclusione è propria).

- Dato un linguaggio regolare L, sappiamo che esiste un DFA $A = (Q, \Sigma, \delta, q_0, F)$ che riconosce L. Inoltre, sappiamo che ogni DFA può essere convertito in una grammatica context-free dove le regole sono del tipo $R_i \to aR_j$ per ogni transizione $\delta(q_i, a) = q_j$ del DFA, e del tipo $R_i \to \varepsilon$ per ogni stato finale del q_i del DFA. Entrambi i tipi di regola rispettano le condizioni di linearità, quindi la grammatica equivalente al DFA è lineare, e questo implica che L è un linguaggio lineare.

- Consideriamo il linguaggio non regolare $L = \{0^n 1^n \mid n \geq 0\}$. La seguente grammatica lineare genera L:

$$S \to 0S1 \mid \varepsilon$$

Quindi, esiste un linguaggio lineare che non è regolare.

Per trasformare un DFA (Automa a Stati Finiti Deterministico) in una grammatica context-free equivalente, si può seguire la seguente procedura:

1. Creare un non-terminale per ogni stato del DFA.
2. Per ogni transizione δ(q, a) = p nel DFA, dove q e p sono stati e a è un simbolo dell'alfabeto, aggiungere una regola della forma: q → ap.
3. Per ogni stato finale f del DFA, aggiungere una regola della forma: f → ε.

La grammatica risultante genererà esattamente lo stesso linguaggio accettato dal DFA di partenza.

Vediamo un esempio pratico:

Consideriamo il seguente DFA che riconosce il linguaggio delle stringhe binarie che terminano con 01:

Dove q0 è lo stato iniziale e q1 è l'unico stato finale.

Applicando la procedura:

1. Creiamo un non-terminale per ogni stato: Q0 e Q1.
2. Per la transizione δ(q0, 0) = q0, aggiungiamo la regola: Q0 → 0Q0.
3. Per la transizione δ(q0, 1) = q1, aggiungiamo la regola: Q0 → 1Q1.
4. Per lo stato finale q1, aggiungiamo la regola: Q1 → ε.

La grammatica risultante è: Q0 → 0Q0 Q0 → 1Q1 Q1 → ε

Che genera esattamente lo stesso linguaggio del DFA di partenza.

Ecco degli esempi per illustrare le quattro tecniche per progettare grammatiche context-free presentate dal professore:

1. Unione di linguaggi più semplici: Supponiamo di voler generare il linguaggio L = L1 ∪ L2, dove L1 = {0^n 1^n | n ≥ 1} e L2 = {0^n 1^m | n ≠ m, n ≥ 1, m ≥ 1}. Possiamo definire due grammatiche separate G1 e G2 per generare rispettivamente L1 e L2, e poi combinarle in una nuova grammatica G che genera L1 ∪ L2.

G1: S → 0A | B A → 0A1 | 1 B → 0B1 | ε

G2: S → 0A | 0B A → 0A1 | 1 B → 0B1 | ε

G: S → S1 | S2 S1 → 0A | B (regole di G1) A → 0A1 | 1 B → 0B1 | ε S2 → 0A | 0B (regole di G2) A → 0A1 | 1 B → 0B1 | ε

2. Trasformare un DFA in CFG: Considera il seguente DFA che riconosce il linguaggio delle stringhe binarie che contengono almeno una sotto-stringa "01":

G: S → S1 | S2 S1 → 0A | B (regole di G1) A → 0A1 | 1 B → 0B1 | ε S2 → 0A | 0B (regole di G2) A → 0A1 | 1 B → 0B1 | ε

2. Trasformare un DFA in CFG: Considera il seguente DFA che riconosce il linguaggio delle stringhe binarie che contengono almeno una sotto-stringa "01":

La grammatica context-free equivalente è: Q0 → 0Q1 | 1Q2 Q1 → 0Q1 | 1Q3 | ε Q2 → 0Q2 | 1Q3 Q3 → 0Q3 | 1Q3 | ε

3. Sottostringhe collegate: Supponiamo di voler generare il linguaggio L = {x#y#z | x, y, z ∈ {0,1}*, |x| = |z|, y è una sotto-stringa di x}. Possiamo definire una grammatica che genera prima una stringa x, poi una sotto-stringa y di x, e infine una stringa z con la stessa lunghezza di x.

S → AxBzC A → 0A | 1A | ε B → 0B0 | 0B1 | 1B0 | 1B1 | ε C → 0C | 1C | ε x → 0x | 1x | ε z → 0z | 1z | ε

4. Strutture ricorsive: Consideriamo il linguaggio L = {0^n 1^m 0^n 1^m | n, m ≥ 1}, che descrive stringhe con due copie identiche di sottostringhe della forma 0^n 1^m. Possiamo definire una grammatica ricorsiva che genera prima una sotto-stringa 0^n 1^m, e poi la ripete.

S → AB A → 0A0 | 0B B → 1B1 | ε

Altro caso particolare:

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.

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

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.

Esercizi PL

2. (12 punti) Considera il linguaggio

$$L_2 = \{1^n w \mid w \text{ è una stringa di 0 e 1 di lunghezza } n\}.$$

Dimostra che L_2 non è regolare.

Soluzione: Usiamo il Pumping Lemma per dimostrare che il linguaggio non è regolare. Supponiamo per assurdo che L_2 sia regolare:

- sia k la lunghezza data dal Pumping Lemma;
- consideriamo la parola $w = 1^k 0^k$, che appartiene ad L_2 ed è di lunghezza maggiore di k;
- sia $w = xyz$ una suddivisione di w tale che $y \neq \varepsilon$ e $|xy| \leq k$;
- poiché $|xy| \leq k$, allora x e y sono entrambe contenute nella sequenza iniziale di 1. Inoltre, siccome $y \neq \varepsilon$, abbiamo che $x = 1^q$ e $y = 1^p$ per qualche $q \geq 0$ e $p > 0$. z contiene la parte rimanente della stringa: $z = 1^{k-q-p}0^k$. Consideriamo l'esponente $i = 0$: la parola xy^0z ha la forma

$$xy^0z = xz = 1^q 1^{k-q-p} 0^k = 1^{k-p} 0^k$$

Poiché $p > 0$, la sequenza iniziale di 1 è più corta della sequenza finale di 0, e quindi la parola iterata xy^0z non può essere scritta nella forma $1^n w$ con $n = |w|$ perché non contiene abbastanza 1 nella parte iniziale.

$$L = \{0^{2n}1^n : n \geq 0\}$$

Soluzione

$$w = 0^{2p}1^p$$

con

$$x = 0^k, y = 0^q, z = 0^{2p-k-q}1^p, p, q \geq 0$$

Si scelga un i tale che $xy^iz \notin L$.

$$xy^iz = 0^k 0^{iq} 0^{2p-k-q} 1^p = 0^{2p+iq-q}$$

Funziona con $i = 2$

$$L = \{0^n 10^n : n \geq 0\}$$

Soluzione

$$w = 0^{2p}1^p$$

con

$$x = 0^p, y = 0^q, z = 0^{p-k-q}10^p, p, q \geq 0$$

Si scelga un i tale che $xy^iz \notin L$.

$$xy^iz = 0^k 0^{iq} 0^{p-k-q} 10^p = 0^{p+iq-q}10^p$$

Funziona con $i = 2$

Esercizi dimostra che L sia regolare:

1. **(12 punti)** Se L è un linguaggio sull'alfabeto $\{0,1\}$, la *rotazione a destra* di L è l'insieme delle stringhe

$$\mathrm{ROR}(L) = \{aw \mid wa \in L, w \in \{0,1\}^*, a \in \{0,1\}\}.$$

Per esempio, se $L = \{0, 001, 10010\}$, allora $\mathrm{ROR}(L) = \{0, 100, 01001\}$. Dimostra che se L è regolare allora anche $\mathrm{ROR}(L)$ è regolare.

Per dimostrare che se L è un linguaggio regolare sull'alfabeto {0, 1}, allora anche la rotazione a destra di L, indicata come ROR(L), è regolare, possiamo utilizzare la proprietà di chiusura dei linguaggi regolari rispetto all'operazione di concatenazione e all'operazione di riflessione.

Definiamo formalmente la rotazione a destra di L:

ROR(L) = {aw | wa ∈ L, w ∈ {0, 1}*, a ∈ {0, 1}}

La rotazione a destra di L genera tutte le stringhe ottenute concatenando una stringa w di L con un singolo simbolo a alla sua sinistra.

Ora, poiché L è un linguaggio regolare, possiamo costruire un DFA M che riconosce L.

Possiamo utilizzare M per costruire un nuovo DFA che riconosce ROR(L). L'idea è di riflettere il DFA M rispetto al suo asse verticale (invertendo le transizioni destra-sinistra) e quindi concatenare una transizione aggiuntiva per ogni stato che porta al nuovo stato iniziale (quello che era lo stato finale di M) con l'input a per ogni a in {0, 1}.

Questo nuovo DFA riconoscerà esattamente le stringhe di ROR(L), dimostrando che ROR(L) è regolare.

In conclusione, se L è un linguaggio regolare, allora anche ROR(L) è regolare.

1. **(12 punti)** Data una stringa w di 0 e 1, il *flip* di w si ottiene cambiando tutti gli 0 in w con 1 e tutti gli 1 in w con 0. Dato un linguaggio L, il flip di L è il linguaggio

$$\mathit{flip}(L) = \{w \in \{0,1\}^* \mid \text{ il flip di } w \text{ appartiene ad } L\}.$$

Dimostra che la classe dei linguaggi regolari è chiusa rispetto all'operazione di flip.

Soluzione: Se L è un linguaggio regolare, allora sappiamo che esiste un DFA $A = (Q, \Sigma, \delta, q_0, F)$ che riconosce L. Costruiamo un DFA $A' = (Q', \Sigma, \delta', q_0', F')$ che accetta il linguaggio $\mathit{flip}(L)$ come segue.

- $Q' = Q$. L'insieme degli stati rimane lo stesso.
- L'alfabeto Σ rimane lo stesso.
- Per ogni stato $q \in Q$, $\delta'(q,0) = \delta(q,1)$ e $\delta'(q,1) = \delta(q,0)$. La funzione di transizione scambia gli 0 con 1 e gli 1 con 0.
- $q_0' = q_0$. Lo stato iniziale non cambia.
- $F' = F$. Gli stati finali rimangono invariati.

Per dimostrare che A' riconosce il linguaggio $\mathit{flip}(L)$, dobbiamo considerare due casi.

- Se $w \in \mathit{flip}(L)$, allora sappiamo che il flip di w appartiene ad L. Chiamiamo $\overline{w}$ il flip di w. Siccome A riconosce L, allora esiste una computazione di A che accetta $\overline{w}$:

$$s_0 \xrightarrow{\overline{w}_1} s_1 \xrightarrow{\overline{w}_2} \ldots \xrightarrow{\overline{w}_n} s_n$$

  con $s_0 = q_0$ e $s_n \in F$. Se scambiamo gli zeri e gli uni nella computazione, otteniamo una computazione accettante per A' sulla parola w. Di conseguenza, abbiamo dimostrato che $w \in L(A')$.
- Viceversa, se w è accettata dal nuovo automa A', allora esiste una computazione accettante che ha la forma

$$s_0 \xrightarrow{w_1} s_1 \xrightarrow{w_2} \ldots \xrightarrow{w_n} s_n$$

  con $s_0 = q_0$, $s_n \in F'$. Se scambiamo gli zeri e gli uni nella computazione, otteniamo una computazione accettante per A sul flip di w. Di conseguenza, il flip di w appartiene ad L e abbiamo dimostrato che $w \in \mathit{flip}(L)$.

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.

Supponiamo che L sia generato dalla grammatica context-free G = (V, Σ, R, S), dove:

- V è l'insieme dei non-terminali
- Σ è l'alfabeto dei terminali
- R è l'insieme delle regole di produzione
- S ∈ V è il non-terminale iniziale

Costruiamo una nuova grammatica context-free G' = (V', Σ, R', S') che genera substring(L) nel seguente modo:

1. V' = V ∪ {S'} ∪ {A, B} (dove A e B sono nuovi non-terminali)
2. R' contiene le seguenti regole:
    - S' → ASB
    - A → aA | ε, per ogni a ∈ Σ
    - B → bB | ε, per ogni b ∈ Σ
    - Tutte le regole di R

La grammatica G' funziona nel seguente modo:

- La regola S' → ASB divide la stringa da generare in tre parti: la prima parte (generata da A) rappresenta la sottostringa u, la seconda parte (generata dalle regole di R) rappresenta la sottostringa v che appartiene a substring(L), e la terza parte (generata da B) rappresenta la sottostringa w.
- Le regole A → aA | ε e B → bB | ε generano tutte le possibili sottostringhe u e w rispettivamente.
- Le regole di R vengono aggiunte direttamente a R' per generare le sottostringhe v appartenenti a L.

Quindi, G' genera esattamente il linguaggio substring(L) = {v | uvw ∈ L per qualche coppia di stringhe u, w}.

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.

Altri esercizi: PDA

3. (12 punti) Mostra che per ogni PDA P esiste un PDA P_2 con due soli simboli di stack tale che $L(P_2) = L(P)$. *Suggerimento:* dai una codifica binaria all'alfabeto di stack di P.

Dato un PDA P = (Q, Σ, Γ, δ, q0, Z0, F), dove Q è l'insieme degli stati, Σ è l'alfabeto di input, Γ è l'alfabeto di stack, δ è la funzione di transizione, q0 è lo stato iniziale, Z0 è il simbolo di stack iniziale e F è l'insieme degli stati finali, possiamo costruire un PDA P2 = (Q', Σ, {0, 1}, δ', q0, 0, F) con due soli simboli di stack {0, 1} nel seguente modo:

1. Q' = Q (gli stati rimangono invariati)
2. Σ rimane invariato
3. L'alfabeto di stack di P2 è {0, 1}
4. La funzione di transizione δ' è definita come segue:
    - Per ogni transizione δ(q, a, X) = (p, γ) in P, dove q, p ∈ Q, a ∈ Σ, X ∈ Γ e γ ∈ Γ*:
        - Codifichiamo X e γ in binario come x_{bin} e y_{bin} rispettivamente
        - Aggiungiamo a δ' le seguenti transizioni: δ'(q, a, x_{bin}) = (p, y_{bin})
    - Per ogni transizione δ(q, a, ε) = (p, γ) in P, dove q, p ∈ Q, a ∈ Σ e γ ∈ Γ*:
        - Codifichiamo γ in binario come γ_{bin}
        - Aggiungiamo a δ' le seguenti transizioni: δ'(q, a, ε) = (p, γ_{bin})
    - Per ogni transizione δ(q, ε, X) = (p, γ) in P, dove q, p ∈ Q, X ∈ Γ e γ ∈ Γ*:
        - Codifichiamo X e γ in binario come X_{bin} e γ_{bin} rispettivamente
        - Aggiungiamo a δ' le seguenti transizioni: δ'(q, ε, X_{bin}) = (p, γ_{bin})

In questa costruzione, abbiamo codificato i simboli di stack di P in una rappresentazione binaria per poterli gestire utilizzando solo i simboli di stack 0 e 1 nel PDA P2. La funzione di transizione δ' è definita in modo da simulare le transizioni di P utilizzando questa codifica binaria.

Possiamo dimostrare che L(P2) = L(P) utilizzando un argomento di simulazione: ogni mossa di P può essere simulata da una sequenza di mosse corrispondenti in P2, e viceversa. Quindi, una stringa w è accettata da P se e solo se è accettata da P2.

Questa dimostrazione mostra che, per ogni PDA P, esiste sempre un PDA P2 con due soli simboli di stack che riconosce lo stesso linguaggio L(P)

1.43 Let A be any language. Define DROP-OUT(A) to be the language containing all strings that can be obtained by removing one symbol from a string in A. Thus, DROP-OUT$(A) = \{xz | xyz \in A \text{ where } x, z \in \Sigma^*, y \in \Sigma\}$. Show that the class of regular languages is closed under the DROP-OUT operation. Give both a proof by picture and a more formal proof by construction as in Theorem 1.47.

Per dimostrare che DROPOUT(A) è un linguaggio regolare, possiamo utilizzare la proprietà di chiusura dei linguaggi regolari rispetto alle operazioni regolari, in particolare l'operazione di proiezione (o cancellazione di simboli).

La dimostrazione procede come segue:

1. Poiché A è un linguaggio regolare, esiste un automa a stati finiti deterministico (AFD) M che riconosce A.
2. Costruiamo un nuovo DFA M' che riconosce DROPOUT(A) a partire da M. M' simulerà M, ma con la possibilità di "saltare" (cancellare) un simbolo in qualsiasi momento durante l'elaborazione della stringa di input.
3. Il DFA M' è definito come M' = (Q', Σ, δ', q'0, F'), dove:
    - Q' = Q × (Q ∪ {q0}) (gli stati di M' sono coppie di stati di M più un possibile stato di partenza)
    - q'0 = (q0, q0) (lo stato iniziale di M' è la coppia formata dallo stato iniziale di M e dallo stato iniziale di M)
    - F' = F × Q (gli stati finali di M' sono le coppie formate da uno stato finale di M e uno stato qualsiasi di M)
    - δ' è definita come segue:
        - δ'((q, p), a) = (δ(q, a), δ(p, a)) per ogni q, p ∈ Q, a ∈ Σ
            1. (se non vengono cancellati simboli, M' simula M)
        - δ'((q, p), ε) = (q, δ(p, ε)) per ogni q ∈ Q, p ∈ Q ∪ {q0}
            1. (M' può cancellare un simbolo passando al prossimo stato di M a partire dal secondo stato della coppia)
4. È possibile dimostrare che L(M') = DROPOUT(A) utilizzando un ragionamento per induzione sulla lunghezza delle stringhe.

Poiché M' è un DFA, DROPOUT(A) è un linguaggio regolare.

Dimostriamo che L(M') = DROPOUT(A) per induzione sulla lunghezza delle stringhe.

Base dell'induzione: Consideriamo le stringhe di lunghezza 0.

- La stringa vuota ε appartiene a DROPOUT(A) se e solo se ε appartiene ad A (poiché non ci sono simboli da cancellare).
- Dall'altro lato, M' accetta ε se e solo se (q0, q0) è uno stato finale di M', cioè q0 è uno stato finale di M. Quindi M' accetta ε se e solo se ε appartiene ad A.

Passo induttivo: Supponiamo che la proprietà valga per tutte le stringhe di lunghezza minore di n, dove n > 0. Dimostriamo che vale anche per le stringhe di lunghezza n.

Sia w = a1a2...an una stringa di lunghezza n, con ai ∈ Σ per 1 ≤ i ≤ n.

(⇒) Supponiamo che w appartenga a DROPOUT(A). Allora esiste una stringa z = a1a2...aj-1ajaj+1...an+1 in A tale che w = a1a2...aj-1aj+1...an+1 (cioè, w è ottenuta da z cancellando il simbolo aj).

Per ipotesi induttiva, la stringa a1a2...aj-1aj+1...an+1 è accettata da M'. Quindi, esiste una sequenza di stati r0, r1, ..., rn in M' tale che r0 = q'0 e rn ∈ F', con δ'(ri-1, ai) = ri per 1 ≤ i ≤ n-1 e δ'(rn-1, an) = rn.

Poiché M' può cancellare un simbolo in qualsiasi momento durante l'elaborazione, significa che esiste un indice k tale che rk-1 = (p, q) e rk = (p', q') con δ'((p, q), ak) = (p', δ(q, ak)) = (p', q').

Quindi, M' può accettare la stringa w seguendo la sequenza di stati r0, r1, ..., rk-1, (p', q'), rk+1, ..., rn, che inizia con q'0, finisce in uno stato finale, e passa attraverso la configurazione (p', q') dopo aver letto i primi k-1 simboli e cancellato ak.

(⇐) Supponiamo che w sia accettata da M'. Allora esiste una sequenza di stati r0, r1, ..., rn in M' tale che r0 = q'0, rn ∈ F', con δ'(ri-1, ai) = ri per 1 ≤ i ≤ n.

Poiché M' può cancellare un simbolo, esiste un indice k tale che rk-1 = (p, q) e rk = (p', q') con δ'((p, q), ak) = (p', δ(q, ak)) = (p', q').

Consideriamo la stringa z = a1a2...ak-1ak+1...an. Poiché rk-1 = (p, q), possiamo vedere che la sequenza di stati p, δ(q, a1), δ(q, a1a2), ..., δ(q, a1a2...ak-1), q', δ(q', ak+1), ..., δ(q', a1a2...ak-1ak+1...an) è una sequenza di stati valida in M che porta M ad accettare z.

Quindi, z appartiene ad A. Inoltre, w = a1a2...ak-1ak+1...an, cioè w è ottenuta da z cancellando il simbolo ak. Pertanto, w appartiene a DROPOUT(A).

Una possibile soluzione usando espressioni regolari qui:

https://scholar.harvard.edu/files/harrylewis/files/2011review_soln.pdf