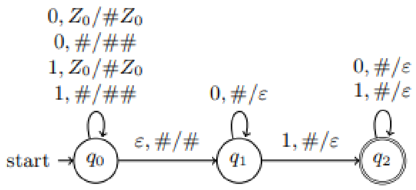
3. Considera l'alfabeto $\Sigma = \{0, 1\}$, e sia L_2 l'insieme di tutte le stringhe che contengono almeno un 1 nella loro seconda metà. Più precisamente, $L_2 = \{uv \mid u \in \Sigma^*, v \in \Sigma^*1\Sigma^* \text{ e } |u| \geq |v|\}$.

(a) Definisci un PDA che riconosce L_2. Spiega perché il PDA riconosce proprio il linguaggio L_2.

(b) Definisci una CFG che genera L_2. Spiega perché la grammatica genera proprio il linguaggio L_2.

3. (a) Il PDA che riconosce L_2 opera nel modo seguente:

- inizia a consumare l'input ed inserisce un carattere # per ogni simbolo che consuma, rimanendo nello stato q_0;
- ad un certo punto, sceglie nondeterministicamente che ha consumato la prima metà della parola, e si sposta nello stato q_1;
- in q_1, estrae un carattere # dalla pila per ogni 0 che consuma dall'input;
- quando legge il primo 1 nella seconda parte della parola, estrae un # dalla pila e si sposta in q_2, che è uno stato finale;
- in q_2, continua ad estrarre un # dalla pila per ogni carattere che consuma (0 o 1).



Per accettare una parola, il PDA deve:

- inserire nella pila un certo numero di #
- estrarre dalla pila un numero di # minore o uguale di quanti ne ha inserito in pila
- consumare almeno un 1 durante lo svuotamento della pila

Quindi l'automa accetta solo parole $w = uv$ dove u è la parte di parola consumata durante la fase di riempimento della pila e v è la parte di parola consumata durante lo svuotamento della pila. La parola v contiene almeno un 1 ed è di lunghezza minore o uguale a u. Se u è più corta di v il PDA svuota la pila prima di riuscire a consumare tutta la parola e si blocca. Se invece v non contiene 1 allora il PDA termina la computazione nello stato q_1 che non è finale.

(b) Per costruire una CFG che genera L_2 prendiamo una qualsiasi parola w che sta in L_2. Se consideriamo l'ultima occorrenza di un 1 nella parola, possiamo riscrivere la parola come $w = u10^k$, con $k \geq 0$ e $|u| \geq k + 1$, perché l'ultima occorrenza di 1 deve stare nella seconda metà della parola. Se spezziamo ulteriormente u in $u = xy$ con $|x| = k + 1$ e $|y| \geq 0$, allora possiamo definire la grammatica che genera L_2 come segue:

$$S \to 0S0 \mid 1S0 \mid 0T1 \mid 1T1$$
$$T \to 0T \mid 1T \mid \varepsilon$$

Nella grammatica, la variabile S genera stringhe del tipo $xT10^k$ con $x \in \{0, 1\}^*$ e $|x| = k + 1$, mentre T genera stringhe $y \in \{0, 1\}^*$ con $|y| \geq 0$. Quindi la grammatica genera tutte e sole le stringhe del tipo $xy10^k$ dove $|xy| \geq k + 1$, che corrispondono alle stringhe che stanno nel linguaggio L_2.

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.

Per costruire un PDA P2 con due soli simboli di stack equivalente a P, usiamo una codifica binaria dei simboli di stack di P.

Sia Γ = {γ1, ..., γn} l'alfabeto di stack di P. Codifichiamo ogni γi come una stringa binaria bi di lunghezza ⌈log2 n⌉.

P2 simula P usando due simboli di stack 0 e 1:

- Quando P fa push di γi, P2 fa push della codifica binaria bi
- Quando P fa pop di γi, P2 fa pop di ⌈log2 n⌉ simboli e verifica che corrispondano a bi
- P2 mantiene gli stessi stati di P

P2 accetta se e solo se P accetta, quindi L(P2) = L(P).

Questa costruzione mostra che è possibile simulare qualsiasi PDA usando solo due simboli di stack, mantenendo lo stesso linguaggio riconosciuto.

2. (9 punti) Chiamiamo k-PDA un automa a pila dotato di k pile. In particolare, uno 0-PDA è un NFA e un 1-PDA è un PDA convenzionale. Sappiamo già che gli 1-PDA sono più potenti degli 0-PDA (nel senso che riconoscono una classe più ampia di linguaggi. Mostra che i 2-PDA sono più potenti degli 1-PDA.

Mostreremo che esiste un linguaggio riconoscibile da un 2-PDA ma non da un 1-PDA.

Consideriamo il linguaggio L = {a^n b^n c^n | n ≥ 0}.

1. L è riconoscibile da un 2-PDA: Un 2-PDA può riconoscere L come segue:
   - Usa la prima pila per contare le 'a'
   - Usa la seconda pila per contare le 'b'
   - Confronta le 'c' con entrambe le pile
2. L non è riconoscibile da un 1-PDA: Supponiamo per assurdo che esista un 1-PDA P che riconosce L. Sia k il numero di stati di P. Consideriamo la stringa w = a^m b^m c^m con m > k. Durante la lettura di a^m b^m, P deve memorizzare informazioni su m nella sua pila. Ma dopo aver letto b^m, P non può conservare abbastanza informazioni per verificare c^m. Usando il pumping lemma per linguaggi context-free, possiamo dimostrare che se P accetta L, allora accetterebbe anche stringhe non in L, come a^m b^m c^(m+1).

Quindi, L è riconoscibile da un 2-PDA ma non da un 1-PDA, dimostrando che i 2-PDA sono più potenti.

3. (9 punti) Chiamiamo k-PDA un automa a pila dotato di k pile. In particolare, uno 0-PDA è un NFA e un 1-PDA è un PDA convenzionale. Sappiamo già che gli 1-PDA sono più potenti degli 0-PDA (nel senso che riconoscono una classe più ampia di linguaggi). Mostra che i 2-PDA sono equivalenti alle Turing Machine.

- 2-PDA ⊆ TM: un 2-PDA può essere simulato da una TM che usa

una traccia del nastro per ogni pila e lo stato finito.

Quindi i 2-PDA sono al più potenti quanto le TM.

- TM $\subseteq$ 2-PDA: Una TM può essere simulata da un 2-PDA che usa:
  - Una pila per simulare il nastro sinistro della TM
  - Una pila per simulare il nastro destro
  - Uno stato finito per tenere traccia dello stato e del simbolo sotto la testina

  Il 2-PDA può simulare ogni mossa della TM con un numero finito di mosse.

  Quindi i 2-PDA sono almeno potenti quanto le TM.

Combinando i due punti, i 2-PDA e le TM sono equivalenti.