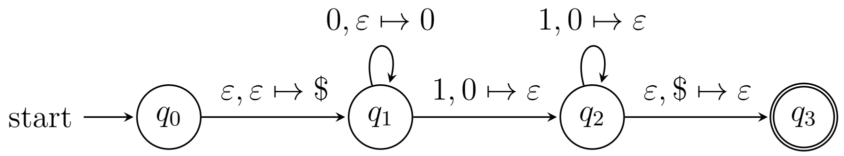
1 Trasformazione del PDA in grammatica

Prendiamo il seguente esempio:

Trasformiamo il PDA per il linguaggio $\{0^n 1^n \mid n \geq 0\}$ in grammatica:



1.1 Analisi del PDA

Prima di procedere con la trasformazione, analizziamo brevemente il PDA dato:

- L'automa a pila riconosce il linguaggio $\{0^n 1^n \mid n \geq 0\}$
- Il PDA ha 4 stati: q_0 (iniziale), q_1, q_2 e q_3 (finale)
- In q_1, per ogni '0' letto, viene inserito un '0' nello stack
- In q_2, per ogni '1' letto, viene rimosso un '0' dallo stack
- Il PDA accetta se, dopo aver letto tutti i simboli di input, raggiunge q_3 svuotando lo stack

1.2 Trasformazione formale in CFG

Per trasformare un PDA in una grammatica context-free, usiamo l'algoritmo standard basato sui tripli di stati. Per ogni coppia di stati q_i e q_j e ogni simbolo di stack X, definiamo un non-terminale A_{ij}^X che genera tutte le stringhe w tali che, partendo da q_i con solo X in cima allo stack, il PDA può raggiungere q_j con lo stack esattamente come era prima di X, consumando esattamente w.

Formalmente, per il nostro PDA:

1. Definiamo i non-terminali A_{ij}^X per ogni $i, j \in \{0, 1, 2, 3\}$ e ogni $X \in \{\$, 0\}$
2. Il simbolo iniziale della grammatica sarà $S = A_{03}^{\$}$, che rappresenta tutte le stringhe che portano dal primo all'ultimo stato, consumando il simbolo di fondo pila
3. Applichiamo le regole di trasformazione per ogni transizione del PDA

Seguendo l'algoritmo, otteniamo la seguente grammatica:

$$
\begin{aligned}
&A^{\$}_{03} \to A^{\$}_{13} \quad \text{(dalla transizione } q_0 \to q_1 \text{ con } \varepsilon, \varepsilon \mapsto \$\text{)}\\
&A^{\$}_{13} \to A^{\$}_{12}A^{\$}_{23} \quad \text{(decomposizione in sottoproblemi)}\\
&A^{\$}_{13} \to \varepsilon \quad \text{(caso } \varepsilon\text{, se } n = 0\text{)}\\
&A^{\$}_{12} \to 0A^{0}_{11} \quad \text{(dalla transizione } q_1 \to q_1 \text{ con } 0, \varepsilon \mapsto 0\text{)}\\
&A^{0}_{11} \to 0A^{0}_{11}A^{0}_{11} \mid \varepsilon \quad \text{(per concatenare più '0')}\\
&A^{\$}_{23} \to \varepsilon \quad \text{(dalla transizione } q_2 \to q_3 \text{ con } \varepsilon, \$ \mapsto \varepsilon\text{)}\\
&A^{0}_{12} \to 1 \quad \text{(dalla transizione } q_1 \to q_2 \text{ con } 1, 0 \mapsto \varepsilon\text{)}\\
&A^{0}_{22} \to 1 \quad \text{(dalla transizione } q_2 \to q_2 \text{ con } 1, 0 \mapsto \varepsilon\text{)}
\end{aligned}
$$

Possiamo semplificare questa grammatica attraverso sostituzioni e eliminazioni di produzioni ridondanti:

$$
\begin{aligned}
S &\to A\\
A &\to 0A1 \mid \varepsilon
\end{aligned}
$$

Questa grammatica genera esattamente il linguaggio $\{0^n1^n \mid n \geq 0\}$.

1.3 Passi dettagliati della trasformazione

Vediamo in dettaglio come applicare l'algoritmo di trasformazione da PDA a CFG:

1. Per ogni tripla (q_i, X, q_j), creiamo un non-terminale A^X_{ij}
2. Per ogni transizione del PDA, aggiungiamo una produzione alla grammatica

Consideriamo le transizioni del nostro PDA:

Transizione	**Da**	**A**	**Input, Stack**	**Nuovo Stack**
T_1	q_0	q_1	ε, ε	$\$$
T_2	q_1	q_1	$0, \varepsilon$	0
T_3	q_1	q_2	$1, 0$	ε
T_4	q_2	q_2	$1, 0$	ε
T_5	q_2	q_3	$\varepsilon, \$$	ε

Da queste transizioni, deriviamo le produzioni della grammatica:

1. Da T_1: Se andiamo da q_0 a q_1 inserendo \$, otteniamo $A^{\$}_{03} \to A^{\$}_{13}$
2. Da T_2: Se in q_1 leggiamo '0' e inseriamo '0' nello stack, poi continuiamo a elaborare, otteniamo $A^X_{1j} \to 0A^0_{1k}A^X_{kj}$ per ogni j, k, X
3. Da T_3 e T_4: Se in q_1 o q_2 leggiamo '1' e rimuoviamo '0' dallo stack, otteniamo $A^0_{12} \to 1$ e $A^0_{22} \to 1$
4. Da T_5: Se da q_2 a q_3 rimuoviamo \$, otteniamo $A^{\$}_{23} \to \varepsilon$

Inoltre, aggiungiamo le produzioni per gestire le transizioni "vuote", ovvero quelle che non consumano input e non modificano lo stack:

$$A_{ii}^X \to \varepsilon \quad \text{per ogni stato } i \text{ e simbolo di stack } X$$

E le produzioni per decomporre il problema in sottoproblemi:

$$A_{ij}^X \to A_{ik}^X A_{kj}^\varepsilon \quad \text{per ogni } i, j, k, X$$

Dopo aver applicato queste regole e semplificato la grammatica (rimuovendo i non-terminali inutili e le produzioni ridondanti), arriviamo alla forma semplificata:

$$\begin{aligned} S &\to A \\ A &\to 0A1 \mid \varepsilon \end{aligned}$$

1.4 Verifica della correttezza

Per verificare che la grammatica ottenuta generi effettivamente il linguaggio $\{0^n 1^n \mid n \geq 0\}$, consideriamo alcune derivazioni:

1. Per $n = 0$, generiamo ε:

$$\begin{aligned} S &\Rightarrow A \\ &\Rightarrow \varepsilon \end{aligned}$$

2. Per $n = 1$, generiamo 01:

$$\begin{aligned} S &\Rightarrow A \\ &\Rightarrow 0A1 \\ &\Rightarrow 0\varepsilon 1 \\ &\Rightarrow 01 \end{aligned}$$

3. Per $n = 2$, generiamo 0011:

$$\begin{aligned} S &\Rightarrow A \\ &\Rightarrow 0A1 \\ &\Rightarrow 00A11 \\ &\Rightarrow 00\varepsilon 11 \\ &\Rightarrow 0011 \end{aligned}$$

In generale, per ogni $n \geq 0$, la grammatica genera esattamente la stringa $0^n 1^n$, come richiesto.

2 Conclusione

Abbiamo trasformato con successo il PDA che riconosce il linguaggio $\{0^n 1^n \mid n \geq 0\}$ in una grammatica context-free equivalente. La grammatica risultante, dopo la semplificazione, è:

$$\begin{aligned} S &\to A \\ A &\to 0A1 \mid \varepsilon \end{aligned}$$

Questa grammatica è semplice ed elegante, e genera esattamente il linguaggio desiderato.