

# Automati a pila

## 1 Automi a pila

Una volta introdotta la classe dei linguaggi context-free, generati dalle CFG, si vuole individuare un modello che permetta di riconoscere tali linguaggi. I riconoscitori trattati finora, cioè gli automi a stati finiti (in tutte le loro varianti), non sono adeguati a questo scopo, poiché si è visto che esistono linguaggi context-free, come ad esempio quello dei palindromi, che non sono regolari, ovvero non sono riconoscibili da automi a stati finiti. L'idea che consente di definire un modello più "potente" è quella di eliminare uno dei grossi vincoli degli automi a stati finiti: la quantità di memoria finita. A tale scopo, si estendono gli automi a stati finiti con una memoria esterna potenzialmente infinita. Tuttavia, perché sia possibile riconoscere esattamente i linguaggi context-free, bisogna imporre su tale memoria una politica di accesso *LIFO*, *Last In First Out*, cioè la memoria deve essere uno *stack*, una *pila*, nella quale si può manipolare solo l'elemento in cima. Il modello risultante è definito formalmente in seguito:

Un **automa a pila** (**PDA**, *PushDown Automaton*) è una settupla

$$P = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F \rangle$$

in cui:

- $Q$  è un insieme finito e non vuoto di **stati**;
- $\Sigma$  è l'**alfabeto di input**;
- $\Gamma$  è l'**alfabeto di stack**;
- $q_0 \in Q$  è lo **stato iniziale**;
- $Z_0 \in \Gamma$  è il **simbolo iniziale di stack**;
- $F \subseteq Q$  è l'insieme degli **stati finali**;
- $\delta$  è la **funzione di transizione**,

$$\delta : Q \times \Sigma_\epsilon \times \Gamma \rightarrow \mathcal{P}_{\text{fin}}(Q \times \Gamma^*)$$

dove la notazione  $\mathcal{P}_{\text{fin}}(S)$  indica l'insieme dei sottoinsiemi finiti di  $S$ .

Quello appena definito è un automa non deterministico con  $\epsilon$ -mosse, un'estensione degli  $\epsilon$ -NFA:<sup>1</sup> dalla definizione di questi ultimi rimangono invariati gli elementi  $Q$ ,  $\Sigma$ ,  $q_0$  e  $F$ .

Lo stack viene modellato come una stringa di simboli di un apposito alfabeto  $\Gamma$ . Per convenzione, il simbolo più a sinistra della stringa è quello in cima allo stack, cioè una stringa  $a_1 \dots a_n$  corrisponde al seguente stack:

$$\begin{array}{|c} a_1 \\ \vdots \\ a_n \end{array}$$

Nelle computazioni, sarà necessario che l'automata riconosca quando lo stack è vuoto. Invece di rappresentare lo stack vuoto con la stringa  $\epsilon$ , risulterà più comodo richiedere che nell'alfabeto di stack sia presente un simbolo distinto  $Z_0 \in \Gamma$ , che si troverà sempre e solo in fondo allo stack:

$$\begin{array}{|c} a_1 \\ \vdots \\ a_n \\ Z_0 \end{array}$$

quando l'automata, guardando lo stack, vedrà  $Z_0$  come elemento in cima (ovvero come unico elemento, poiché è anche in fondo), saprà che lo stack è (di fatto) vuoto.

L'ultimo elemento della definizione è la funzione di transizione  $\delta$ , che descrive come l'automata evolve in una mossa di computazione. Un'applicazione di questa funzione ha la forma

$$(q, a, X) \xrightarrow{\delta} \{(p_1, \gamma_1), \dots, (p_n, \gamma_n)\}$$

dove  $q, p_1, \dots, p_n \in Q$ ,  $a \in \Sigma_\epsilon$ ,  $X \in \Gamma$  e  $\gamma_1, \dots, \gamma_n \in \Gamma^*$ . Come negli  $\epsilon$ -NFA, l'evoluzione dipende dallo stato attuale  $q$  e dal simbolo in input  $a$  (che può essere  $\epsilon$  per rappresentare una transizione "spontanea", senza consumo di input), ma in più viene considerato anche il simbolo  $X$  in cima allo stack, quindi il dominio di  $\delta$  è  $Q \times \Sigma_\epsilon \times \Gamma$ . Trattandosi poi di un automa non deterministico, il risultato di  $\delta$  è un insieme di possibili evoluzioni, ciascuna delle quali è una coppia  $(p_i, \gamma_i) \in Q \times \Gamma^*$ , dove  $p_i$  è lo stato in cui passa l'automata e  $\gamma_i$  è una stringa che descrive come modificare lo stack.  $\Gamma^*$ , e di conseguenza  $Q \times \Gamma^*$ , è un insieme infinito, dunque il codominio di  $\delta$  non può essere l'insieme delle parti  $2^{Q \times \Gamma^*}$ , perché ciò ammetterebbe come risultati anche sottoinsiemi infiniti, sui quali non sarebbe possibile definire le computazioni; per considerare invece solo i sottoinsiemi finiti, si sceglie come codominio  $\mathcal{P}_{\text{fin}}(Q \times \Gamma^*)$ .

La stringa  $\gamma_i$  indica i simboli con cui sostituire  $X$  in cima allo stack:

---

<sup>1</sup>Si possono definire anche i PDA deterministici e i PDA non deterministici senza  $\epsilon$ -mosse, ma la variante con  $\epsilon$ -mosse è la più comoda nelle dimostrazioni, e per brevità le altre non verranno presentate.

- se  $\gamma_i = X$ , lo stack rimane immutato;

$$\begin{array}{|c} X \\ Y \\ \vdots \end{array} \xrightarrow{\gamma_i = X} \begin{array}{|c} X \\ Y \\ \vdots \end{array}$$

- se  $\gamma_i = \epsilon$ , si elimina il simbolo  $X$  dalla cima dello stack;

$$\begin{array}{|c} X \\ Y \\ \vdots \end{array} \xrightarrow{\gamma_i = \epsilon} \begin{array}{|c} Y \\ \vdots \end{array}$$

- se infine  $\gamma_i = \alpha Z$ , con  $\alpha \in \Gamma^{*2}$  e  $Z \in \Gamma$ ,  $X$  viene sostituito con  $Z$  e i simboli di  $\alpha$  vengono aggiunti uno a uno in cima allo stack, procedendo da destra verso sinistra, quindi se  $\alpha = a_1 \dots a_n$  rimane in cima allo stack il simbolo  $a_1$ .

$$\begin{array}{|c} X \\ Y \\ \vdots \end{array} \xrightarrow{\gamma_i = a_1 \dots a_n Z} \begin{array}{|c} a_1 \\ \vdots \\ a_n \\ Z \\ Y \\ \vdots \end{array}$$

## 2 Esempio: palindromi di lunghezza pari

Si consideri il linguaggio dei palindromi di lunghezza pari,

$$L_{ww^R} = \{ww^R \mid w \in \{0,1\}^*\}$$

cioè il linguaggio tale che  $\alpha \in L_{ww^R}$  se e solo se esiste  $w \in \{0,1\}^*$  tale che  $\alpha = ww^R$ .  
Esso è generato dalla CFG con le seguenti regole di produzione:

$$P \rightarrow \epsilon \mid 0P0 \mid 1P1$$

Si vuole costruire un PDA  $P_{ww^R}$  che riconosca (intuitivamente) il linguaggio  $L_{ww^R}$ . Siccome ogni stringa del linguaggio è formata da due “metà”,  $w$  e  $w^R$ , una strategia per il riconoscimento è memorizzare nello stack i simboli della prima metà, poi estrarre i simboli dallo stack e verificare che corrispondano a quelli della seconda metà della stringa (sfruttando l’inversione dell’ordine dei simboli che viene determinata naturalmente dalla politica di accesso LIFO). C’è però un problema: il PDA può leggere solo un simbolo alla volta, quindi non ha modo di sapere quanto è arrivato a metà della stringa di input.

<sup>2</sup>Volendo, si potrebbe imporre che  $\alpha \in \Gamma \cup \{\epsilon\}$ , ovvero limitare la “crescita” dello stack a un simbolo alla volta: il modello risultante sarebbe equivalente, ma meno comodo per le dimostrazioni.

Una soluzione è sfruttare il non determinismo, facendo a ogni simbolo due “scommesse”, di essere e di non essere arrivati a metà, e lasciando che la computazione data dalla scommessa errata si blocchi.

Per implementare la strategia appena descritta,  $P_{ww^R}$  deve avere tre stati ( $q_0$ ,  $q_1$  e  $q_2$ ):

- l'automa si trova nello stato iniziale  $q_0$  finché non assume di aver terminato la lettura della prima metà della stringa,  $w$ ;
- l'automa si trova in  $q_1$  quando sta verificando che la seconda metà della stringa sia  $w^R$ ;
- l'automa si trova nello stato finale  $q_2$  quando ha riconosciuto una stringa della forma  $ww^R$ , che potrebbe essere l'intera stringa in input  $\alpha = ww^R$ , oppure solo un suo prefisso (se  $\alpha = ww^R\beta$ ), e in quest'ultimo caso la computazione si dovrebbe bloccare alla lettura del simbolo successivo, per evitare l'accettazione di una stringa non palindroma.

In base alle osservazioni fatte finora, si può iniziare a dare una caratterizzazione formale dell'automa:

$$P_{ww^R} = \langle \{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, Z_0\}, \delta, q_0, Z_0, \{q_2\} \rangle$$

- gli stati sono i tre appena descritti, con  $q_0$  iniziale e  $q_2$  finale;
- l'alfabeto di input è  $\{0, 1\}$ , quello su cui è costruito il linguaggio da riconoscere;
- l'alfabeto di stack è  $\{0, 1, Z_0\}$ , perché lo stack deve poter contenere i simboli della stringa in input e il simbolo di fine stack  $Z_0$ .

Rimane da definire la funzione di transizione  $\delta$ , il “programma” dell'automa:

- Nello stato  $q_0$ , se si assume di non essere arrivati a metà, bisogna leggere un simbolo di input, memorizzarlo nello stack, e rimanere in  $q_0$ . L'azione da compiere è indipendente dagli specifici simboli in input e in cima allo stack, ma formalmente va specificata per ogni combinazione di tali simboli:

$$\begin{aligned} \delta(q_0, 0, Z_0) &= \{(q_0, 0Z_0)\} & \delta(q_0, 0, 0) &= \{(q_0, 00)\} & \delta(q_0, 0, 1) &= \{(q_0, 01)\} \\ \delta(q_0, 1, Z_0) &= \{(q_0, 1Z_0)\} & \delta(q_0, 1, 0) &= \{(q_0, 10)\} & \delta(q_0, 1, 1) &= \{(q_0, 11)\} \end{aligned}$$

In sintesi, si potrebbe scrivere:

$$\forall a \in \{0, 1\}, \forall X \in \{0, 1, Z_0\} \quad \delta(q_0, a, X) = \{(q_0, aX)\}$$

Se invece si assume di essere arrivati a metà della stringa, cioè di aver completato la lettura della parte  $w$ , e dover invece iniziare la verifica della parte  $w^R$ , si passa allo stato  $q_1$  senza consumare input (si fa un' $\epsilon$ -mossa) e senza modificare lo stack:

$$\delta(q_0, \epsilon, Z_0) = \{(q_1, Z_0)\} \quad \delta(q_0, \epsilon, 0) = \{(q_1, 0)\} \quad \delta(q_0, \epsilon, 1) = \{(q_1, 1)\}$$

ovvero

$$\forall X \in \{0, 1, Z_0\} \quad \delta(q_0, \epsilon, X) = \{(q_1, X)\}$$

- Nello stato  $q_1$ , se si legge un simbolo in input  $a$  che coincide con il simbolo in cima allo stack, si cancella  $a$  dalla cima dello stack e si resta in  $q_1$ :

$$\delta(q_1, 0, 0) = \{(q_1, \epsilon)\} \quad \delta(q_1, 1, 1) = \{(q_1, \epsilon)\}$$

Se invece  $a$  non coincide con la cima dello stack, eventualmente anche perché lo stack è vuoto, la computazione si blocca:

$$\begin{aligned} \delta(q_1, 0, 1) &= \emptyset & \delta(q_1, 0, Z_0) &= \emptyset \\ \delta(q_1, 1, 0) &= \emptyset & \delta(q_1, 1, Z_0) &= \emptyset \end{aligned}$$

A ogni passo in  $q_1$ , l'automa prova anche ad assumere di essere arrivato alla fine dell'input. Uno stack vuoto significa allora che per ogni simbolo di  $w$  è stato letto un corrispondente simbolo nella seconda metà  $w^R$ , quindi la stringa in input è palindroma (o meglio, lo è il prefisso letto finora, che l'automa suppone sia l'intera stringa), e si passa allo stato finale  $q_2$ :

$$\delta(q_1, \epsilon, Z_0) = \{(q_2, Z_0)\}$$

Se invece lo stack non è vuoto, significa che mancano ancora dei simboli per completare  $w^R$ , dunque l'automa non può assumere (per ora) che l'input sia un palindromo, e la computazione relativa a tale assunzione si blocca:

$$\delta(q_1, \epsilon, 0) = \emptyset \quad \delta(q_1, \epsilon, 1) = \emptyset$$

- Nello stato  $q_2$ , l'automa ha riconosciuto un palindromo di lunghezza pari, che suppone sia l'intera stringa di input. Da qui non si può procedere ulteriormente,

$$\forall a \in \{0, 1, \epsilon\}, \forall X \in \{0, 1, Z_0\} \quad \delta(q_2, a, X) = \emptyset$$

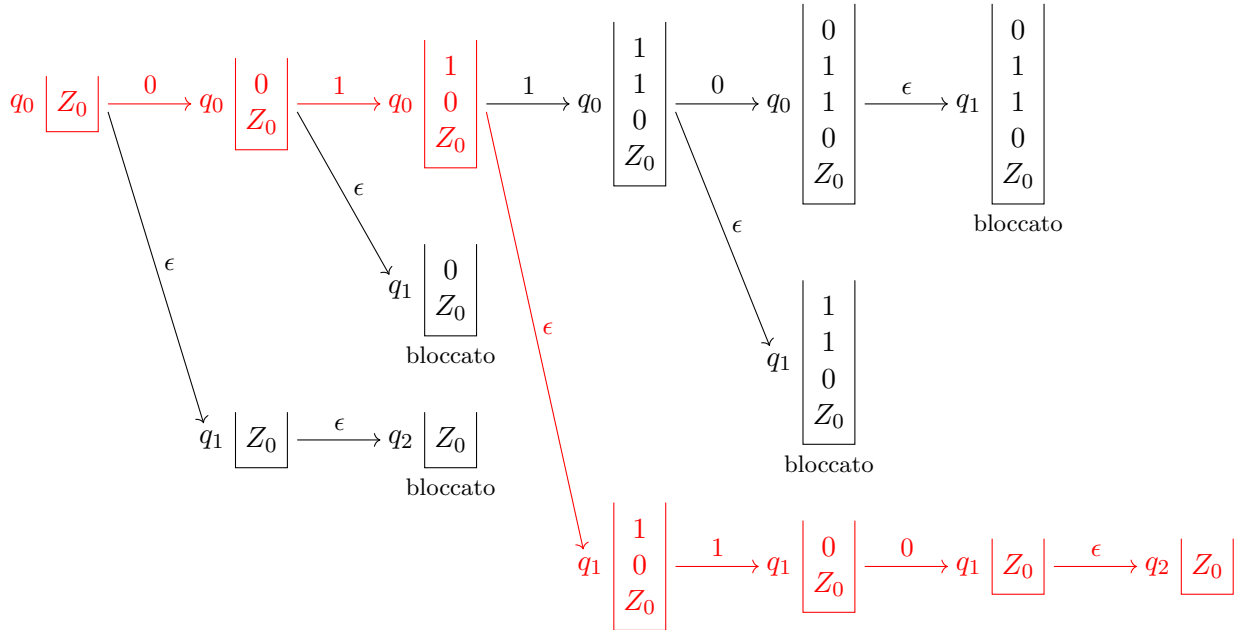
quindi la stringa viene accettata se non ci sono più simboli in input, altrimenti la computazione si blocca, perché il palindromo riconosciuto era solo un prefisso dell'intero input.

Riassumendo, la funzione di transizione è definita come:

$$\begin{aligned} \forall a \in \{0, 1\}, \forall X \in \{0, 1, Z_0\} \quad & \delta(q_0, a, X) = \{(q_0, aX)\} \\ \forall X \in \{0, 1, Z_0\} \quad & \delta(q_0, \epsilon, X) = \{(q_1, X)\} \\ \\ \delta(q_1, 0, 0) &= \{(q_1, \epsilon)\} & \delta(q_1, 1, 1) &= \{(q_1, \epsilon)\} \\ \delta(q_1, 0, 1) &= \emptyset & \delta(q_1, 0, Z_0) &= \emptyset \\ \delta(q_1, 1, 0) &= \emptyset & \delta(q_1, 1, Z_0) &= \emptyset \\ \delta(q_1, \epsilon, Z_0) &= \{(q_2, Z_0)\} & \delta(q_1, \epsilon, 0) &= \delta(q_1, \epsilon, 1) = \emptyset \\ \\ \forall a \in \{0, 1, \epsilon\}, \forall X \in \{0, 1, Z_0\} \quad & \delta(q_2, a, X) = \emptyset \end{aligned}$$

## 2.1 Esempio di computazione

In seguito è raffigurato l'albero di computazione dell'automa  $P_{ww^R}$  sulla stringa 0110, con evidenziato in rosso il percorso che porta all'accettazione:



## 3 Rappresentazione grafica

I PDA possono essere rappresentati graficamente, in modo simile agli automi a stati finiti. L'unica cosa che cambia sono le etichette delle transizioni: se  $(p, \gamma) \in \delta(q, a, X)$ , lo stato di partenza  $p$  e lo stato di arrivo  $q$  sono indicati dalla freccia della transizione, ma rimangono da indicare in input  $a$  (come negli automi a stati finiti), il simbolo  $X$  in cima allo stack e la stringa  $\gamma$  di modifica dello stack. La notazione che viene usata per questi tre elementi è  $a, X/\gamma$ : in pratica, la parte prima della barra indica quando si applica la transizione, mentre la parte dopo indica come viene modificato lo stack quando l'automa esegue questa transizione.

Ad esempio, l'automa  $P_{ww^R}$  descritto prima è rappresentato dal seguente diagramma di transizione:

