Automi e CFG

Automi e RG

Sia D = (S, Σ, move, s_0, F) un automa a stati finiti e G_D = (Σ, V, S, P) la CFG così definita :

- Se S = {s_0,s_1, …, s_n}, introduciamo un non-terminale V_i per ogni stato s_i in S. Quindi V={V_0,V_1, … , V_n}
- S=V_0 il non terminale iniziale è quello associato ad s_0
- L'insieme delle produzioni P è costruito come segue:
  - se move(s_i, a) = s_j aggiungi in P la produzione $V_i \rightarrow a\ V_j$
  - se move(s_i, a) = s_j e $s_j \in F$ aggiungi in P anche la produzione $V_i \rightarrow a$

Automi e RG

- $G_D = (\Sigma, V, S, P)$ è una grammatica regolare (Regular Grammar – RG) e genera lo stesso linguaggio riconosciuto da D, i.e. $L(D)=L(G_D)$
- Gli automi a stati finiti (FA – Finite Automaton) sono in grado di riconoscere solamente una particolare classe di linguaggi, detti **linguaggi regolari** (gli stessi denotati da un'espressione regolare)
- Consideriamo il linguaggio $L = \{a^n b^n \mid n \geq 0\}$.
- L è libero da contesto; è generato dalla seguente CFG:
  - $S \rightarrow aSb \mid \varepsilon$
- Ma non è regolare – non esiste alcun automa a stati finito in grado di riconoscere L

Automi e RG

Da cosa deriva questa mancanza di espressività?

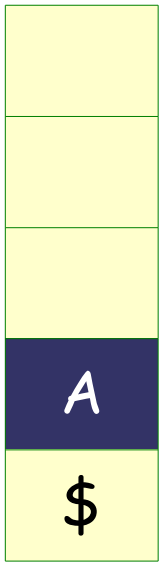
- Il numero degli stati è fissato a priori e non consente di trattare quei problemi che richiedono un conteggio senza un limite prefissato
- Es: un FA con 4 stati può riconoscere aabb ma non aaabbb
- Rendere il numero degli stati infinito è impraticabile
- Soluzione: estendere gli automi a stati finiti con una struttura dati, nello specifico una **pila**, per memorizzare delle informazioni

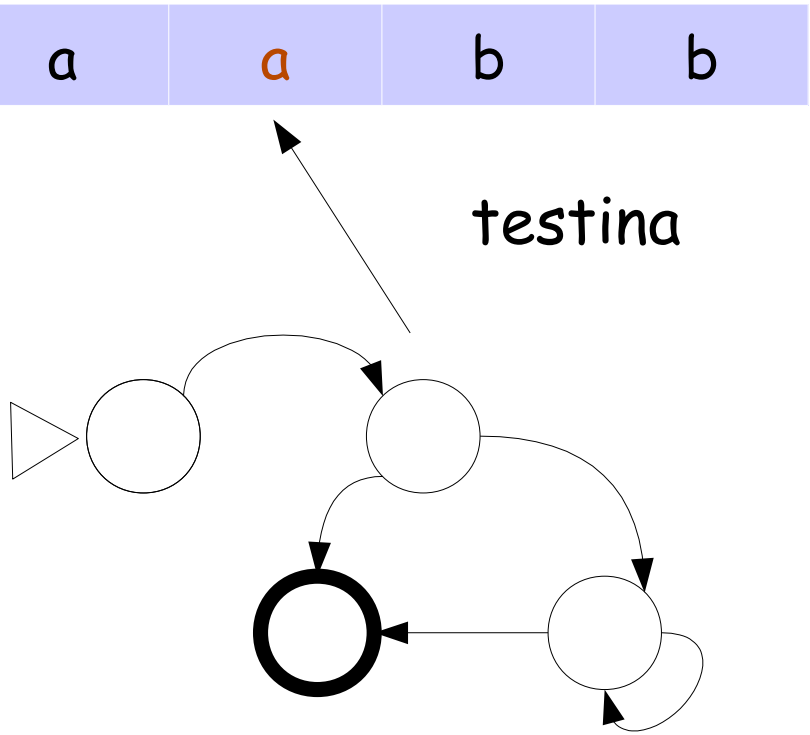
AUTOMI A PILA (PDA)

Un PDA è composto da:

- Un controllo a stati finiti (stati + funzione di transizione)
- Un input
- Una pila

PDA è un acronimo per Pushdown Automaton





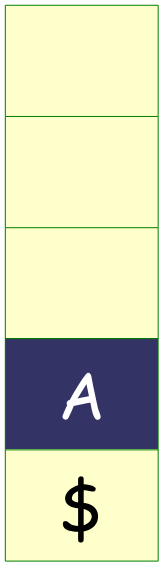
AUTOMI A PILA (PDA)

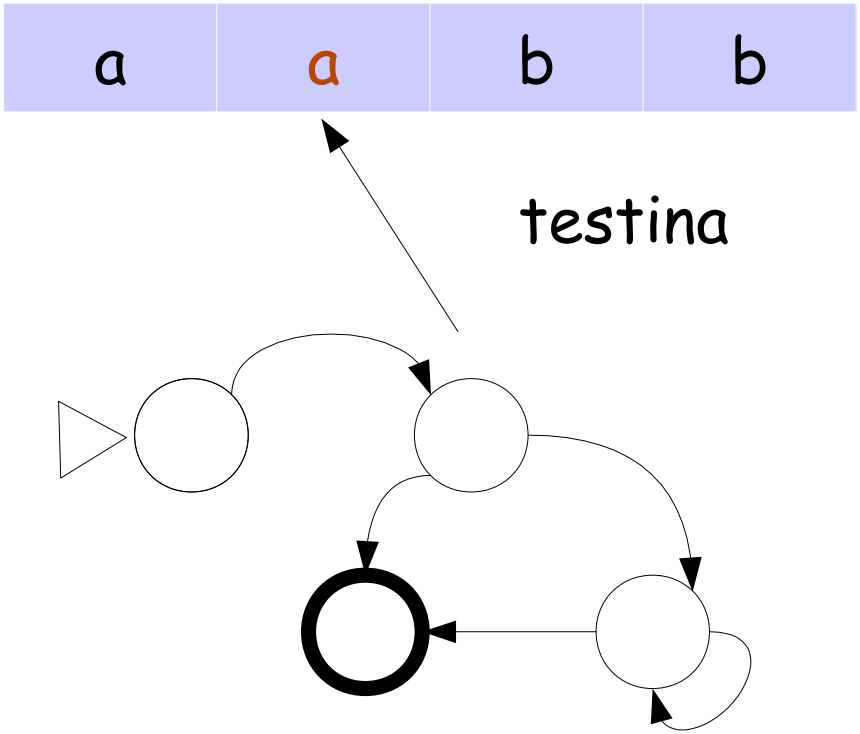
Legge

- un simbolo dal nastro di input (e avanza la testina di una posizione) ed
- Il simbolo in testa alla pila

Cambia stato e

- aggiorna il contenuto dello stack





PDA: definizione

Un **automa a pila (PDA) non deterministico** è una tupla

$P = (S, \Sigma, \Gamma, \delta, s_0, \$, F)$ dove:

- S è un insieme finito di stati, ed $s_0 \in S$ è lo stato iniziale
- Σ è un alfabeto finito di simboli in input
- Γ è un alfabeto finito di simboli dello stack (t.c. $\Sigma \cap \Gamma = \varnothing$)
- $\delta: S \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow p(S \times \Gamma^*)$ è una funzione di transizione
- $\$ \in F$ è il simbolo iniziale dello stack
- $F \subseteq Q$ è l'insieme degli stati finali

PDA: definizione

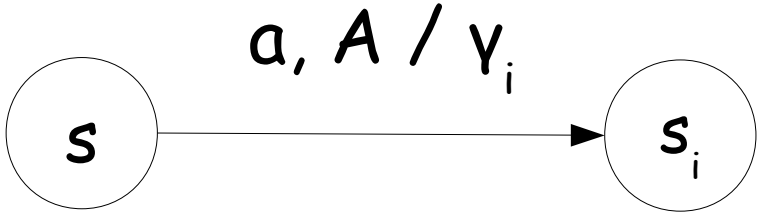
Per ogni $s \in Q$, $a \in Q\ (\Sigma \cup \{\varepsilon\})$ e $A \in \Gamma$:

$$\delta(s, a, A) = \{ \langle s_1, \gamma_1 \rangle, \langle s_2, \gamma_2 \rangle, \ldots, \langle s_n, \gamma_n \rangle \}$$

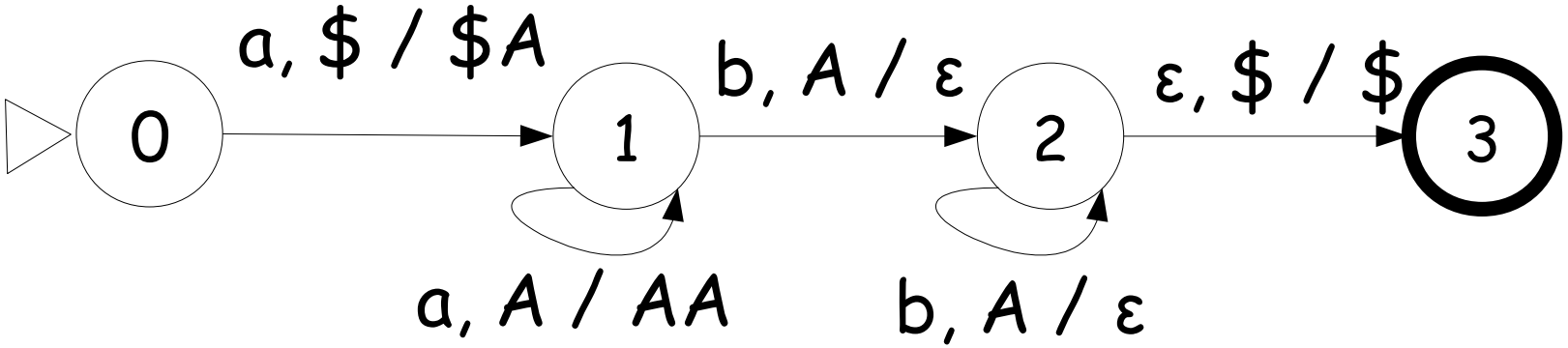
- È un insieme finito di coppie della forma $\langle s_i, \gamma_i \rangle$ dove s_i è uno stato e γ_i è una stringa sull'alfabeto Γ
- Il simbolo A (al top dello stack) viene rimosso e sostituito γ_i
- Il nuovo simbolo in cima allo stack è l'ultimo simbolo di γ_i (ES: se $\gamma_i = AB$, B è il nuovo simbolo al top dello stack)
- N. B. $\gamma_i = \varepsilon$, questa operazione corrisponde ad una pop(A)

Rappresentazione grafica

- Molto simile a quella degli automi a stati finiti
- Una transizione $\langle s_i, \gamma_i \rangle \in \delta(s, a, A)$ viene rappresentata come:



PDA: un esempio



Fa una serie di push(A) - mentre legge una a (dal nastro in input) seguite da delle pop() – quando legge delle b.
Effettua una transizione nello stato finale (3) solo se ha terminato di leggere la stringa in input e lo stack è vuoto

$\delta(2, \varepsilon, \$) = \{<3, \$>\}$

PDA: configurazioni

Sia $P = (S, \Sigma, \Gamma, \delta, s_0, \$, F)$ un PDA.

- Una **configurazione** di P è una tupla (s, w, γ) dove:
  - $s \in S$ è uno stato
  - $w \in \Sigma^*$ rappresenta la porzione di stringa in input ancora da leggere
  - $\gamma \in \Gamma^*$ è il contenuto dello stack
- Configurazioni iniziali: della forma $(s_0, w, \$)$
- Configurazioni finali: quelle della forma (s, ε, γ) con $s \in F$

PDA: computazioni

Sia $P = (S, \Sigma, \Gamma, \delta, s_0, \$, F)$ un PDA. Definiamo:

- **Passo** è un modo per determinare la prossima configurazione a partire da quella corrente (in base alla funzione di transizione)
  - se $(s, aw, \gamma A)$ è una configurazione e $(s', \gamma') \in \delta(s, a, A)$ allora $(s, aw, \gamma A) \Rightarrow (s', w, \gamma\gamma')$
- **Computazione**: è una sequenza di zero o più passi della forma:

$$(s_0, w, \$) \Rightarrow (s_1, w_1, \gamma_1) \Rightarrow (s_2, w_2, \gamma_2) \Rightarrow \ldots \Rightarrow (s_n, w_n, \gamma_n)$$

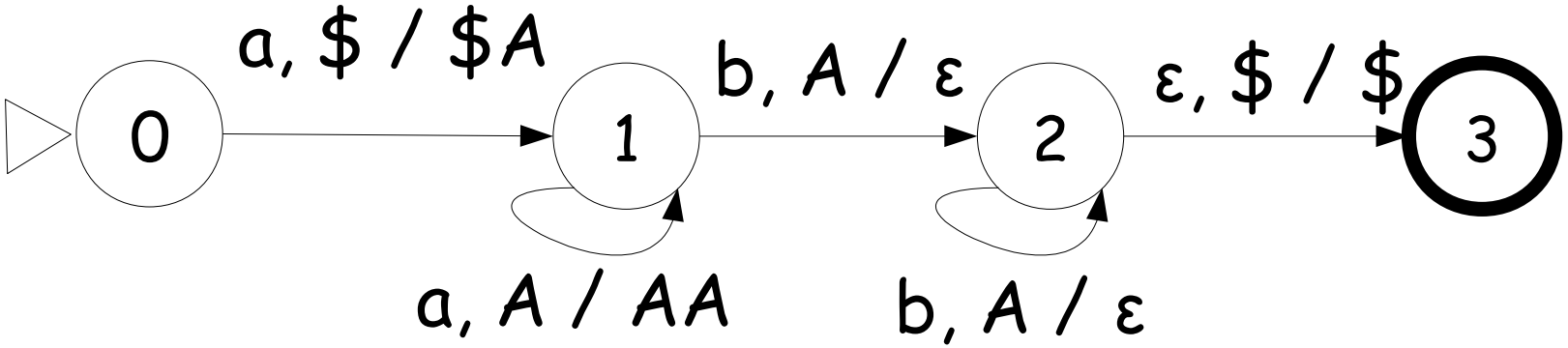
- scriviamo che $(s_0, w, \$) \overset{*}{\Rightarrow} (s_n, w_n, \gamma_n)$

PDA: linguaggio accettato

Sia $P = (S, \Sigma, \Gamma, \delta, s_0, \$, F)$ un PDA. Il linguaggio accettato da P è definito come segue:

$$L(P) = \{ w \in \Sigma^* \mid (s_0, w, \$) \stackrel{*}{\Rightarrow} (s_n, \varepsilon, \gamma) \text{ e } s_n \in F \}$$

PDA: un esempio



$(0, a^n b^n, \$) \Rightarrow$

$(1, a^{n-1} b^n, \$A) \Rightarrow (1, a^{n-2} b^n, \$AA) \Rightarrow \ \ldots \ \Rightarrow (1, b^n, \$A^n) \Rightarrow$

$(2, b^{n-1}, \$A^{n-1}) \Rightarrow (2, b^{n-2}, \$A^{n-2}) \Rightarrow \ \ldots \ \Rightarrow (2, \varepsilon, \$) \Rightarrow$

$(3, \varepsilon, \$)$

$$L(P) = \{a^n b^n \mid n \geq 1\}$$

Le parole palindrome

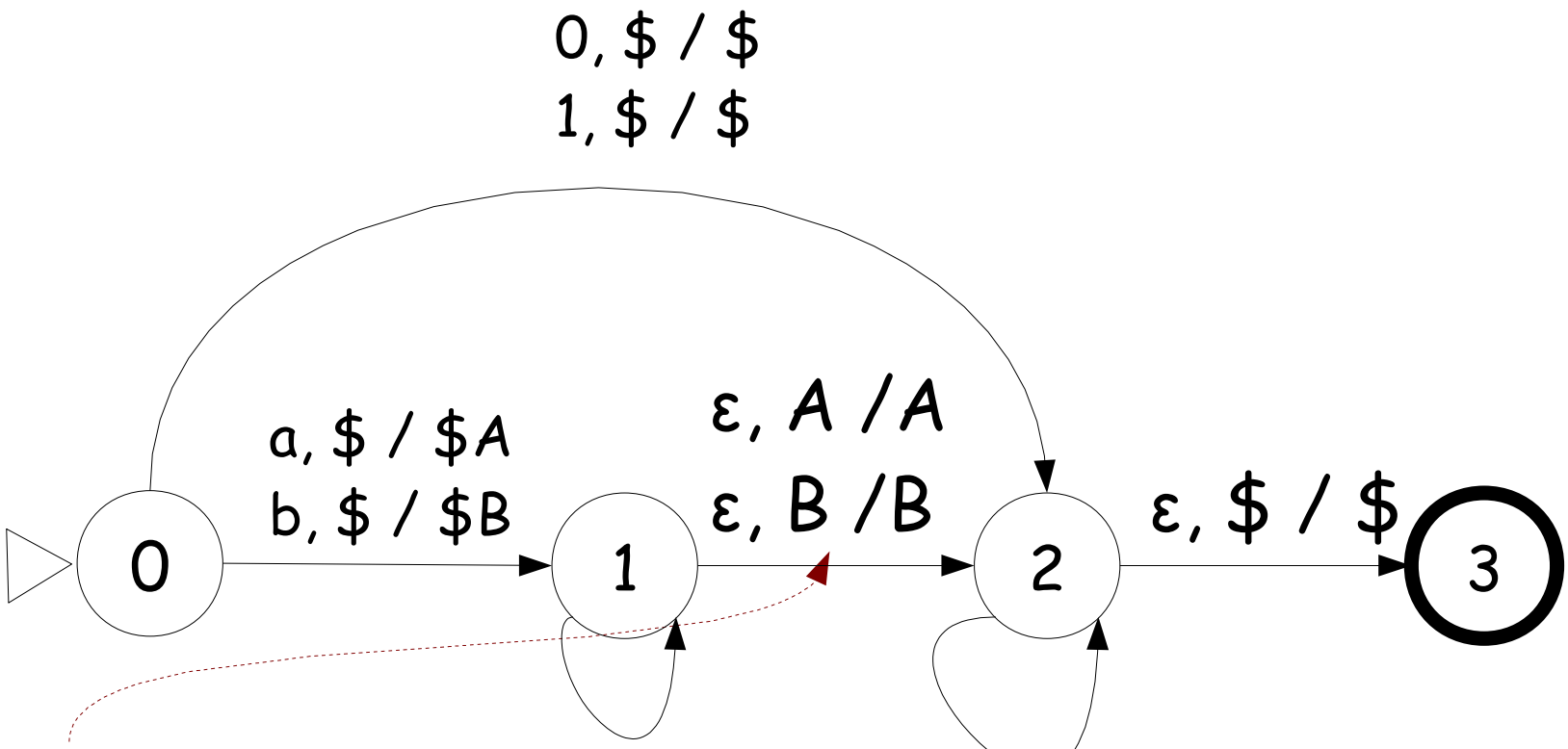
Sia L il linguaggio delle stringhe palindrome generato dalla seguente grammatica

$$S \rightarrow aSa \mid bSb \mid \epsilon$$

$L = \{w\, w^R \mid w \in \{a,b\}^*\}$

L'automa a pila in grado che accetta L è il seguente

Le parole palindrome



Abbiamo raggiunto il centro della stringa
Evolviamo nello stato 2 e cominciamo ad verificare che quello che rimane sia il reverse di quanto memorizzato nella pila

PDA: definizione

Un PDA **deterministico** è una tupla $P = (S, \Sigma, \Gamma, \delta, s_0, \$, F)$ dove:

- S è un insieme finito di stati, ed $s_0 \in S$ è lo stato iniziale
- Σ è un alfabeto finito di simboli in input
- Γ è un alfabeto finito di simboli dello stack (t.c. $\Sigma \cap \Gamma = \emptyset$)
- $\delta: S \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow p(S \times \Gamma^*)$ è una funzione di transizione
- $\$ \in F$ è il simbolo iniziale dello stack
- $F \subseteq Q$ è l'insieme degli stati finali

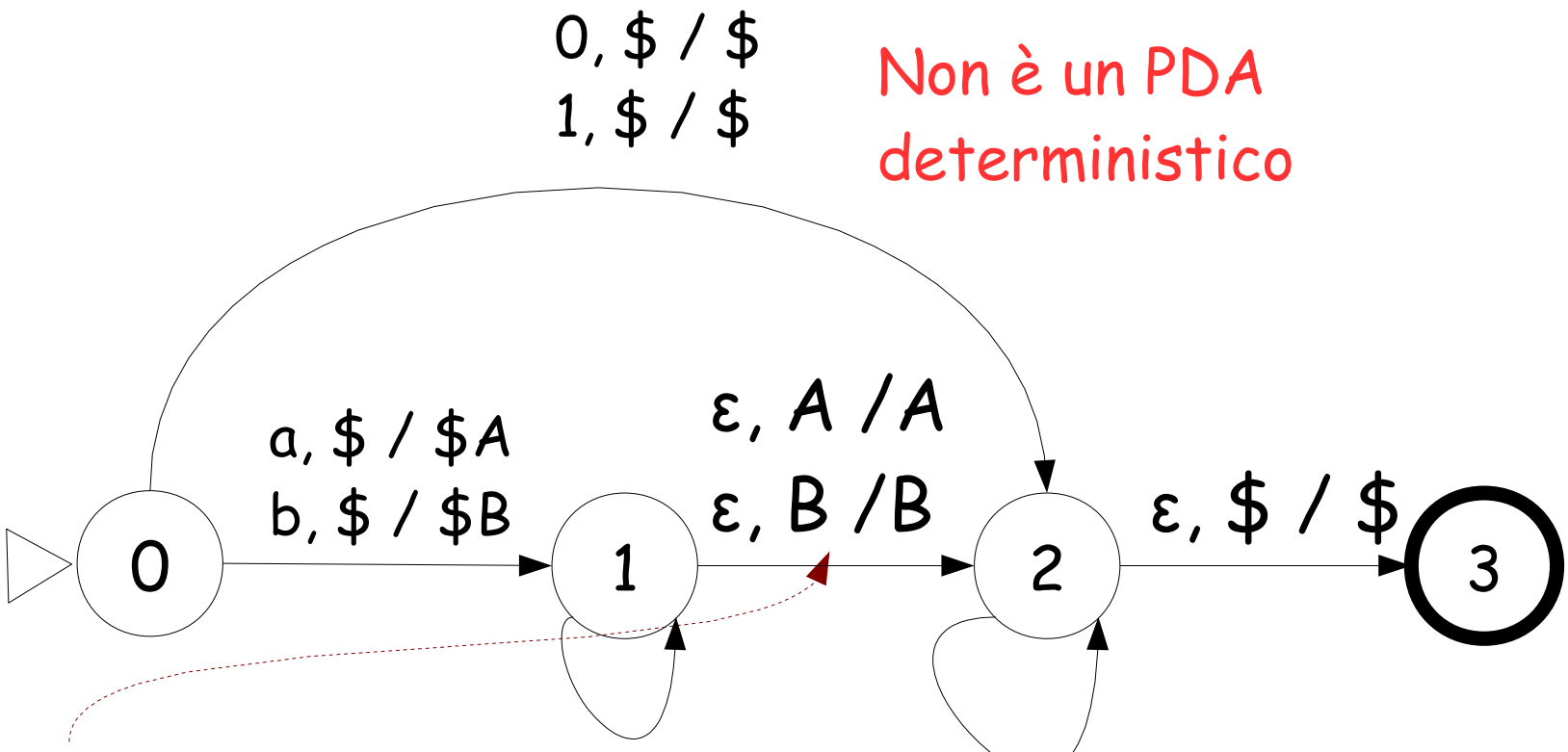
Imponiamo alcuni vincoli sulla funzione di transizione

PDA: definizione

Un PDA **deterministico** è una tupla $P = (S, \Sigma, \Gamma, \delta, s_0, \$, F)$ dove:

- ....
- δ: $S \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow p(S \times \Gamma^*)$ è una funzione di transizione tale che:
  - Per ogni $s \in S$, $a \in \Sigma \cup \{\varepsilon\}$ e $A \in \Gamma$, $|\delta(s, a, A)| \leq 1$
  - Per ogni $s \in S$ e $A \in \Gamma$, se $\delta(s, \varepsilon, A) \neq \emptyset$ (è definito) allora $\delta(s, a, A) = \emptyset$ (non è definito) per ogni $a \in \Sigma$
  - ....

Le parole palindrome



Abbiamo raggiunto il centro della stringa
Evolviamo nello stato 2 e cominciamo ad verificare che quello che rimane sia il reverse di quanto memorizzato nella pila

Le parole palindrome

Non esiste alcun PDA deterministico in grado di accettare $L = \{w\, w^R \mid w \in \{a,b\}^*\}$

I PDA non deterministici sono più espressivi di quelli Deterministici (accettano una più ampia classe di linguaggi)

I PDA non deterministici accettano la classe dei linguaggi Liberi da contesto

PDA e parsing

PDA e derivazioni leftmost

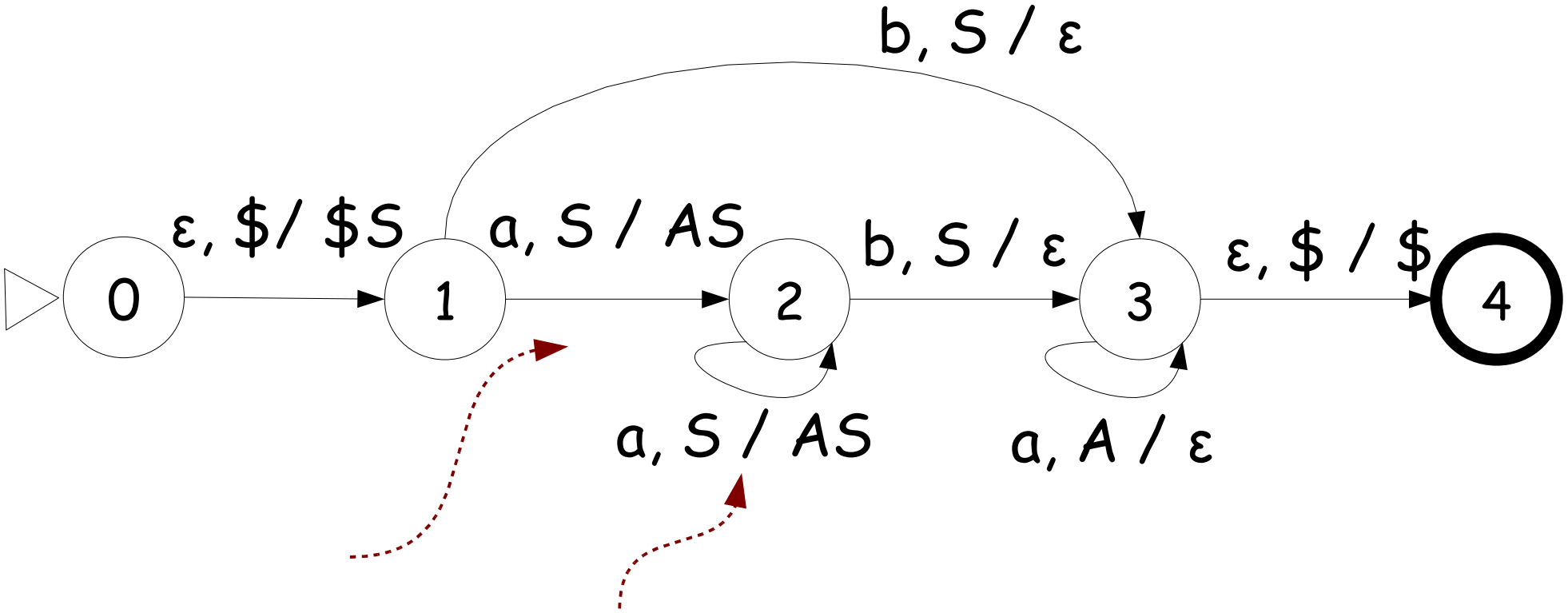
- Consideriamo la CFG **S → aSa | b**
- $L(G) = \{a^n\ b\ a^n \mid n \geq 0\}$
- Ed una derivazione leftmost della stringa **w = aabaa**

$S \Rightarrow aSa \Rightarrow aaSaa \Rightarrow aabaa$

Come possiamo usare un PDA per simulare tale derivazione?

PDA e derivazioni leftmost

$S \Rightarrow aSa \Rightarrow aaSaa \Rightarrow aabaa$



Aggiungiamo:

- una A per ricordarci di aver letto una a dall'input nastro in input
- ed una S perché S è prossimo non terminale da riscrivere