Logica e Modelli Computazionali

Automi a Pila

Marco Console

Ingegneria Informatica e Automatica (Sapienza, Università di Roma)

Capacità e Limiti dei Linguaggi Regolari

- **Definizione**. Un **linguaggio $\mathcal{L}$ è detto regolare** se esiste un **ASFD A tale che $L(A) = \mathcal{L}$**
  - Un linguaggio è regolare se e solo se esiste un ASFD che lo riconosce

- **Teorema 1.** Il linguaggio $\mathrm{L}(A)$ riconosciuto da un ϵ-ASFND A è **regolare**
- **Corollario.** Il linguaggio $\mathrm{L}(A)$ riconosciuto da un ASFND A è **regolare**
  - Non-determinismo e ϵ-transizioni non sono sufficiente a definire macchine più potente

- **Teorema 2.** Esiste almeno un linguaggio $\mathcal{L}$ che **non è regolare**
  - Tutti quelli che non rispettano la proprietà definita dal **Pumping Lemma**
  - **Esempio**. Parentesi ben formate, ad esempio

- **Intuitivamente**, quello agli ASFD mancano due caratteristiche
  1. La possibilità di leggere più volte lo stesso simbolo dell'input e ..
  2. **Una forma più evoluta di memoria**

Automi (con Memoria) a Pila

Computazione di un Automa con Memoria – Esempio

- Supponiamo di voler riconoscere il linguaggio delle parentesi ben formate
  - **Definizione**. Una stringa s sull'alfabeto $\Sigma = \{(,)\}$ è una stringa di parentesi ben formata se
    - $s = (\ )$ oppure
    - $s = (\ p\ )$ e p è una stringa ben formata

- Potremmo implementare il seguente algoritmo
  1. **Leggi** il prossimo carattere c della stringa dell'input
     1. **Se** c è il simbolo (**Allora** metti c in memoria
     2. **Se** c è il simbolo) **Allora** rimuovi un simbolo dalla memoria
  2. **Se** non ci sono più simboli nell'input da leggere **Allora**
     1. **Se** la memoria è vuota **Allora** accetta l'input
     2. **Altrimenti** rigetta l'input

Con quale modello computazionale possiamo implementarlo??

Computazione di un Automa con Memoria – Esempio

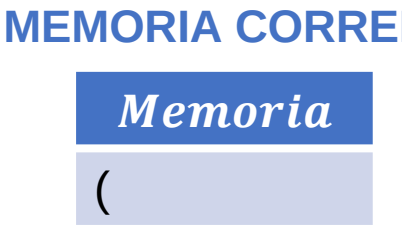
1. **Leggi** il prossimo carattere *c* della stringa dell'input
    1. **Se** *c* è il simbolo (**Allora** metti *c* in memoria
    2. **Se** *c* è il simbolo) **Allora** rimuovi un simbolo dalla memoria
2. **Se** non ci sono più simboli nell'input da leggere **Allora**
    1. **Se** la memoria è vuota **Allora** accetta l'input
    2. **Altrimenti** rigetta l'input

MEMORIA CORRENTE

Memoria

Computazione di un Automa con Memoria – Esempio

1. **Leggi** il prossimo carattere *c* della stringa dell'input
    1. **Se** *c* è il simbolo (**Allora** metti *c* in memoria
    2. **Se** *c* è il simbolo) **Allora** rimuovi un simbolo dalla memoria
2. **Se** non ci sono più simboli nell'input da leggere **Allora**
    1. **Se** la memoria è vuota **Allora** accetta l'input
    2. **Altrimenti** rigetta l'input



Computazione di un Automa con Memoria – Esempio

1. **Leggi** il prossimo carattere *c* della stringa dell'input
    1. **Se** *c* è il simbolo (**Allora** metti *c* in memoria
    2. **Se** *c* è il simbolo) **Allora** rimuovi un simbolo dalla memoria
2. **Se** non ci sono più simboli nell'input da leggere **Allora**
    1. **Se** la memoria è vuota **Allora** accetta l'input
    2. **Altrimenti** rigetta l'input

MEMORIA CORRENTE

Memoria
(
(

Computazione di un Automa con Memoria – Esempio

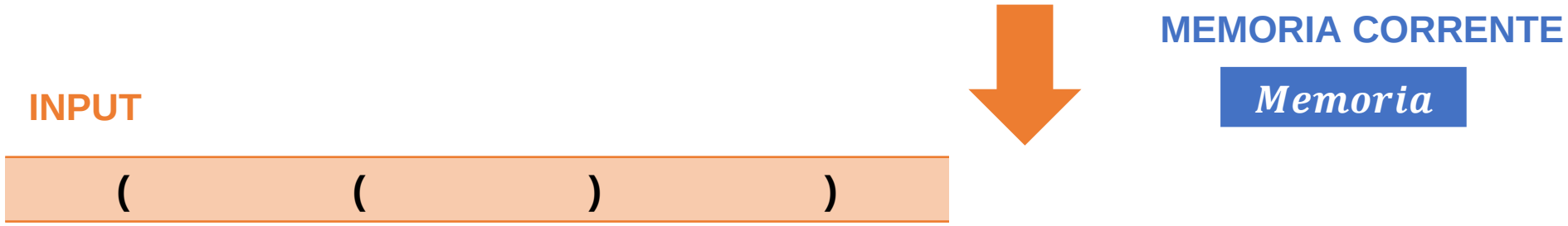
1. **Leggi** il prossimo carattere *c* della stringa dell'input
    1. **Se** *c* è il simbolo (**Allora** metti *c* in memoria
    2. **Se** *c* è il simbolo) **Allora** rimuovi un simbolo dalla memoria
2. **Se** non ci sono più simboli nell'input da leggere **Allora**
    1. **Se** la memoria è vuota **Allora** accetta l'input
    2. **Altrimenti** rigetta l'input

MEMORIA CORRENTE

Memoria
(

Computazione di un Automa con Memoria – Esempio

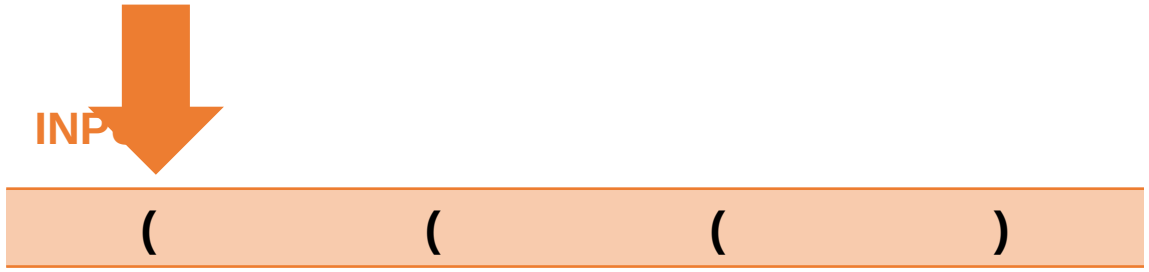
1. **Leggi** il prossimo carattere *c* della stringa dell'input
    1. **Se** *c* è il simbolo (**Allora** metti *c* in memoria
    2. **Se** *c* è il simbolo) **Allora** rimuovi un simbolo dalla memoria
2. **Se** non ci sono più simboli nell'input da leggere **Allora**
    1. **Se** la memoria è vuota **Allora** accetta l'input
    2. **Altrimenti** rigetta l'input



Accetta la Stringa di Input

Computazione di un Automa con Memoria – Esempio

1. **Leggi** il prossimo carattere *c* della stringa dell'input
    1. **Se** *c* è il simbolo (**Allora** metti *c* in memoria
    2. **Se** *c* è il simbolo) **Allora** rimuovi un simbolo dalla memoria
2. **Se** non ci sono più simboli nell'input da leggere **Allora**
    1. **Se** la memoria è vuota **Allora** accetta l'input
    2. **Altrimenti** rigetta l'input



MEMORIA CORRENTE

Memoria

Computazione di un Automa con Memoria – Esempio

1. **Leggi** il prossimo carattere *c* della stringa dell'input
    1. **Se** *c* è il simbolo (**Allora** metti *c* in memoria
    2. **Se** *c* è il simbolo) **Allora** rimuovi un simbolo dalla memoria
2. **Se** non ci sono più simboli nell'input da leggere **Allora**
    1. **Se** la memoria è vuota **Allora** accetta l'input
    2. **Altrimenti** rigetta l'input

MEMORIA CORRENTE

Computazione di un Automa con Memoria – Esempio

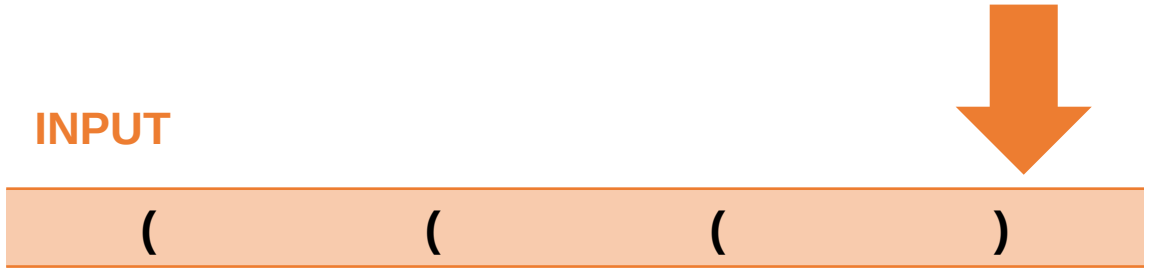
1. **Leggi** il prossimo carattere *c* della stringa dell'input
    1. **Se** *c* è il simbolo (**Allora** metti *c* in memoria
    2. **Se** *c* è il simbolo) **Allora** rimuovi un simbolo dalla memoria
2. **Se** non ci sono più simboli nell'input da leggere **Allora**
    1. **Se** la memoria è vuota **Allora** accetta l'input
    2. **Altrimenti** rigetta l'input

MEMORIA CORRENTE

Memoria
(
(

Computazione di un Automa con Memoria – Esempio

1. **Leggi** il prossimo carattere *c* della stringa dell'input
    1. **Se** *c* è il simbolo (**Allora** metti *c* in memoria
    2. **Se** *c* è il simbolo) **Allora** rimuovi un simbolo dalla memoria
2. **Se** non ci sono più simboli nell'input da leggere **Allora**
    1. **Se** la memoria è vuota **Allora** accetta l'input
    2. **Altrimenti** rigetta l'input



MEMORIA CORRENTE

Memoria
(
(
(

Computazione di un Automa con Memoria – Esempio

1. **Leggi** il prossimo carattere *c* della stringa dell'input
    1. **Se** *c* è il simbolo (**Allora** metti *c* in memoria
    2. **Se** *c* è il simbolo) **Allora** rimuovi un simbolo dalla memoria
2. **Se** non ci sono più simboli nell'input da leggere **Allora**
    1. **Se** la memoria è vuota **Allora** accetta l'input
    2. **Altrimenti** rigetta l'input

INPUT

(	(	(	)

MEMORIA CORRENTE

Memoria
(
(

Computazione di un Automa con Memoria – Esempio

1. **Leggi** il prossimo carattere *c* della stringa dell'input
    1. **Se** *c* è il simbolo (**Allora** metti *c* in memoria
    2. **Se** *c* è il simbolo) **Allora** rimuovi un simbolo dalla memoria
2. **Se** non ci sono più simboli nell'input da leggere **Allora**
    1. **Se** la memoria è vuota **Allora** accetta l'input
    2. **Altrimenti** rigetta l'input

INPUT

(	(	(	)

MEMORIA CORRENTE

Memoria
(
(

Rifiuta la Stringa di Input

Automi (con Memoria) a Pila – Intuizione

- Procediamo ad introdurre un **nuovo modello computazionale** in grado di eseguire la computazione che abbiamo descritto informalmente nei lucidi precedenti

- Le macchine di tale modello continuano **a non poter leggere più volte i simboli della stringa di input**
  - Come per gli Automi a Stati Finiti un simbolo letto viene "consumato" dalla computazione

- Ma queste macchine hanno accesso ad una **forma di memoria strutturata come una Pila**
  - Stack o Last-In First-Out (LIFO) Queue

- La **funzione di transizione** che definisce il comportamento di tali macchine dipende da
  - Lo **stato corrente** della macchina (**come gli ASF**)
  - Il **simbolo corrente** della stringa in input (**come gli ASF**)
  - Il **simbolo affiorante** dalla Pila della memoria (**non presente negli ASF**)

Automa a Pila (Non Deterministico)

- **Definizione**. Un **automa a pila non deterministico** (d'ora in poi semplicemente **automa a pila**) è una 6-tupla M della forma $M =< \Sigma, \Gamma, Q, q_0, F, \delta >$, dove:
  1. Σ è **l'alfabeto di input**
  2. Γ è un insieme finito di simboli, chiamato **l'insieme dei simboli della pila**
  3. Q è un insieme finito e non vuoto di **stati**
  4. $q_0 \in Q$ è lo **stato iniziale**
  5. $F \subseteq Q$ è **l'insieme degli stati finali**
  6. $\delta: Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\}) \rightarrow P\big(Q \times (\Gamma \cup \{\epsilon\})\big)$ è la **funzione di transizione**

Computazione di Automi a Pila – Intuizione

- La definizione degli **Automi a Pila** segue la nostra intuizione di macchine in grado di utilizzare una forma di memoria definita come una Pila

- **Intuizione**. Ad ogni passo, a partire dallo **stato attuale**, dal **carattere letto dalla stringa di input** e dal **carattere affiorante dalla memoria**, l'automa
  1. **Consuma** il **simbolo corrente della stringa di input oppure fa un passo ϵ**
  2. **Sostituisce** il **simbolo affiorante dalla memoria** con un altro simbolo (non necessariamente diverso dall'attuale)
  3. **Modifica** lo **stato attuale** con un nuovo stato (non necessariamente diverso dall'attuale)

- **Intuizione**. Stiamo definendo una **forma di ϵ-ASFND** aggiungendo l'accesso a una Pila

Esecuzioni di un Automa a Pila – Preliminari

- Sia s una stringa sull'alfabeto Σ tale che $\epsilon \notin \Sigma$ e $\mathrm{T} \subseteq \Sigma$ un alfabeto.

- **Definizione**. La **restrizione di s ad Σ'** ($s_{|\mathrm{T}}$) è la stringa ottenuta eliminando da s tutti i simboli $c \notin T$

- **Definizione**. Una **ϵ-estensione** di s è una stringa s' sull'alfabeto $\Sigma \cup \{\epsilon\}$ tale che ${s'}_{|\Sigma} = s$
  - La restrizione di s' su Σ coincide con s
  - In altre parole, s' può aggiungere solamente il simbolo ϵ ad s ma un numero arbitrario di volte

- **Esempio**. La restrizione $s_{|T}$ della stringa $s = "asd"$ su $\Sigma = \{a, s, d\}$ all'alfabeto $T = \{a, d\}$ è la stringa $"ad"$

- **Esempio**. La stringa $s' = "a\epsilon s\epsilon d"$ è una ϵ-estensione di $s = "asd"$ su $\Sigma = \{a, s, d\}$ ${s'}_{|\Sigma} = s$
- **Esempio**. La stringa $s'' = "a\epsilon ss\epsilon d"$ non è una ϵ-estensione di $s = "asd"$ su $\Sigma = \{a, s, d\}$ ${s''}_{|\Sigma} = "assd"$

Esecuzioni di un Automa a Pila

- Siano $M =< \Sigma, \Gamma, Q, I, F, \delta >$ un Automa a Pila e $s =$ "$c_1 c_2 \dots c_n$" $\in \Sigma^*$ una stringa con $|s| = n$

- **Definizione**. Una **esecuzione di A su S** è una sequenza $(q_0, \dots, q_k) \in Q^{k+1}$ di $k+1$ elementi di Q tale che esiste una **ϵ-estensione** "$x_1 x_2 \dots x_k$" di s e una **sequenza di stringhe** $t_1, t_2, \dots, t_k$ sull'alfabeto Γ^* con le seguenti proprietà
  1. $q_1 = I$ **e** $t_1 = \epsilon$ (l'esecuzione parte sempre dallo stato iniziale e dalla memoria vuota)
  2. $(q_{i+1}, g') \in \delta(q_i, k_1, g)$ **con** $t_i = g\,k$ **e** $t_{i+1} = g'k$ **per** $i = 1, \dots, k$ (l'esecuzione è coerente con δ)

- **Definizione**. Una esecuzione di $(q_0, \dots, q_k)$ di A su S è **accettante** se il suo stato finale q_n **è in F**

Esecuzioni di un Automa a Pila – Intuizione

- Una esecuzione è una sequenza di stati che richiede
  1. Una **ϵ-estensione** dell'input coerente (per accomodare i passi ϵ com per gli ϵ-ASFND)
  2. Una **sequenza di stringhe** che rappresentano gli **stati della memoria**

- Il primo di ogni esecuzione è quello iniziale e la prima configurazione della memoria di ogni esecuzione è la stringa vuota
  – La macchina parte "dall'inizio" con memoria inalterata

- Ad ogni passo, la macchina
  – aggiorna lo stato interno,
  – rimuove il simbolo affiorante g dalla pila [`pop`]
  – aggiunge g' [`push` g']

- Se $g = \epsilon$, l'effetto sulla memoria è solo l'aggiunta di g' [`push` g']
- Se $g' = \epsilon$, l'effetto sulla memoria è solo la rimozione del simbolo affiorante g [`pop`]

Linguaggio Riconosciuto da un Automa a Pila

- **Definizione**. Dato un **Automa a Pila** $M =< \Sigma, \Gamma, Q, I, F, \delta >$ e una stringa $x \in \Sigma^*$
  - x è **accettata** da A se **esiste almeno una** esecuzione accettante di A su x
  - Altrimenti, x è **rifiutata**

- **Definizione**. Sia $M =< \Sigma, \Gamma, Q, I, F, \delta >$ un **Automa a Pila**. **Il linguaggio riconosciuto da** M è il linguaggio $L(M)$ sull'alfabeto Σ tale che

$$L(M) = \{x \in \Sigma^* \mid x \text{ è } accettata\ da\ M\}$$

- **Domanda**. È vero che ogni linguaggio riconosciuto da un **Automa a Pila** è regolare?
  - Ovvero, possiamo sempre definire un **ASFD** equivalente a un **Automa a Pila**?
  - Ovvero, l'aggiunta della memoria a pila aumenta davvero il potere computazionale degli automi?

Linguaggi NON Regolari e Automi a Pila

- **Proposizione 1**. Il linguaggio $\mathcal{L} = \{a^m b^m \mid m \geq 1\}$ non è regolare
- **Prova**. Applicando la proprietà definita dal Pumping Lemma (vedi lucidi precedenti)

- **Proposizione 2** . Esiste un Automa a Pila M tale che $L(M) = \mathcal{L}$
- **Prova**. Implementiamo con un Automa a Pila un algoritmo in tre fasi
  1. Fase 1. Leggi a e aggiungi un simbolo alla pila, Se leggi un b passa alla Fase 2.
  2. Fase 2. Leggi b e rimuovi un simbolo dalla pila. Se la stringa finisce o la pila finisce passa alla Fase 3
  3. Fase 3. Se la stringa e la pila sono entrambe vuote accetta, altrimenti rifiuta
- Definiamo il seguente Automa a Pila $\mathrm{M} =< \Sigma, \Gamma, Q, I, F, \delta >$ come segue
- $\Sigma = \{a, b\}$; $\Gamma = \{0,1\}$; $Q = \{q_0, q_1, q_2, q_3, q_4\}$; $I = q_0$; $F = \{q_3\}$
- δ è definita dalla tabella successiva. Ogni cella rappresenta l'insieme $\delta(q, c, g)$ per una qualche combinazione di $q \in Q$, $c \in \Sigma$, $g \in \Gamma$

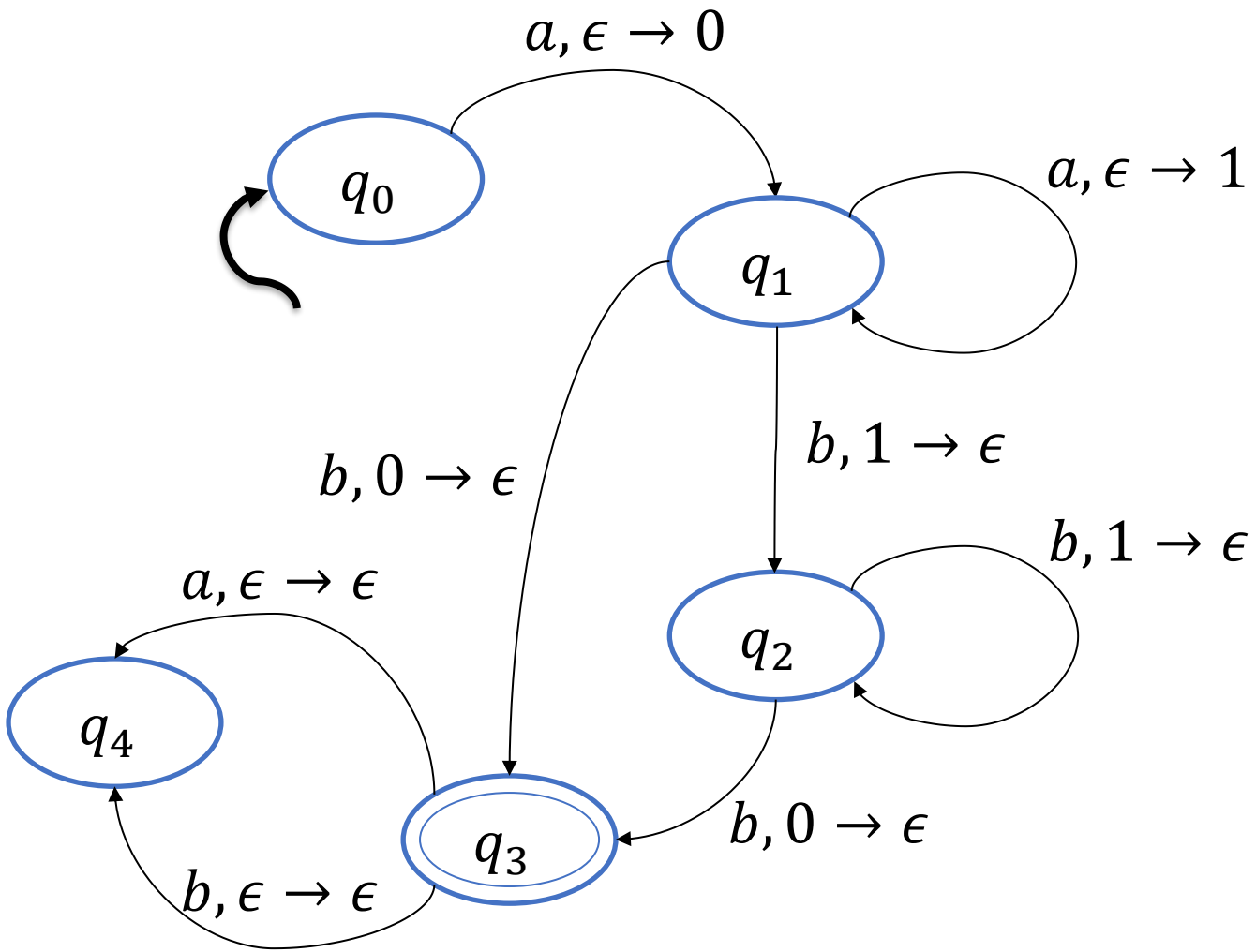
Linguaggi NON Regolari e Automi a Pila

$\delta(q, c, g)$									
q	$c = a;\ g = 0$	$c = a;\ g = 1$	$c = a;\ g = \epsilon$	$c = b;\ g = 0$	$c = b;\ g = 1$	$c = b;\ g = \epsilon$	$c = \epsilon;\ g = 0$	$c = \epsilon;\ g = 1$	$c = \epsilon;\ g = \epsilon$
q_0	$\emptyset$	$\emptyset$	$\{(q_1, 0)\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
q_1	$\emptyset$	$\emptyset$	$\{(q_1, 1)\}$	$\{(q_3, \epsilon)\}$	$\{(q_2, \epsilon)\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
q_2	$\emptyset$	$\emptyset$	$\emptyset$	$\{(q_3, \epsilon)\}$	$\{(q_2, \epsilon)\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
q_3	$\emptyset$	$\emptyset$	$\{(q_4, \epsilon)\}$	$\emptyset$	$\emptyset$	$\{(q_4, \epsilon)\}$	$\emptyset$	$\emptyset$	$\emptyset$
q_4	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Linguaggi NON Regolari e Automi a Pila

Significato $x, y \rightarrow z$

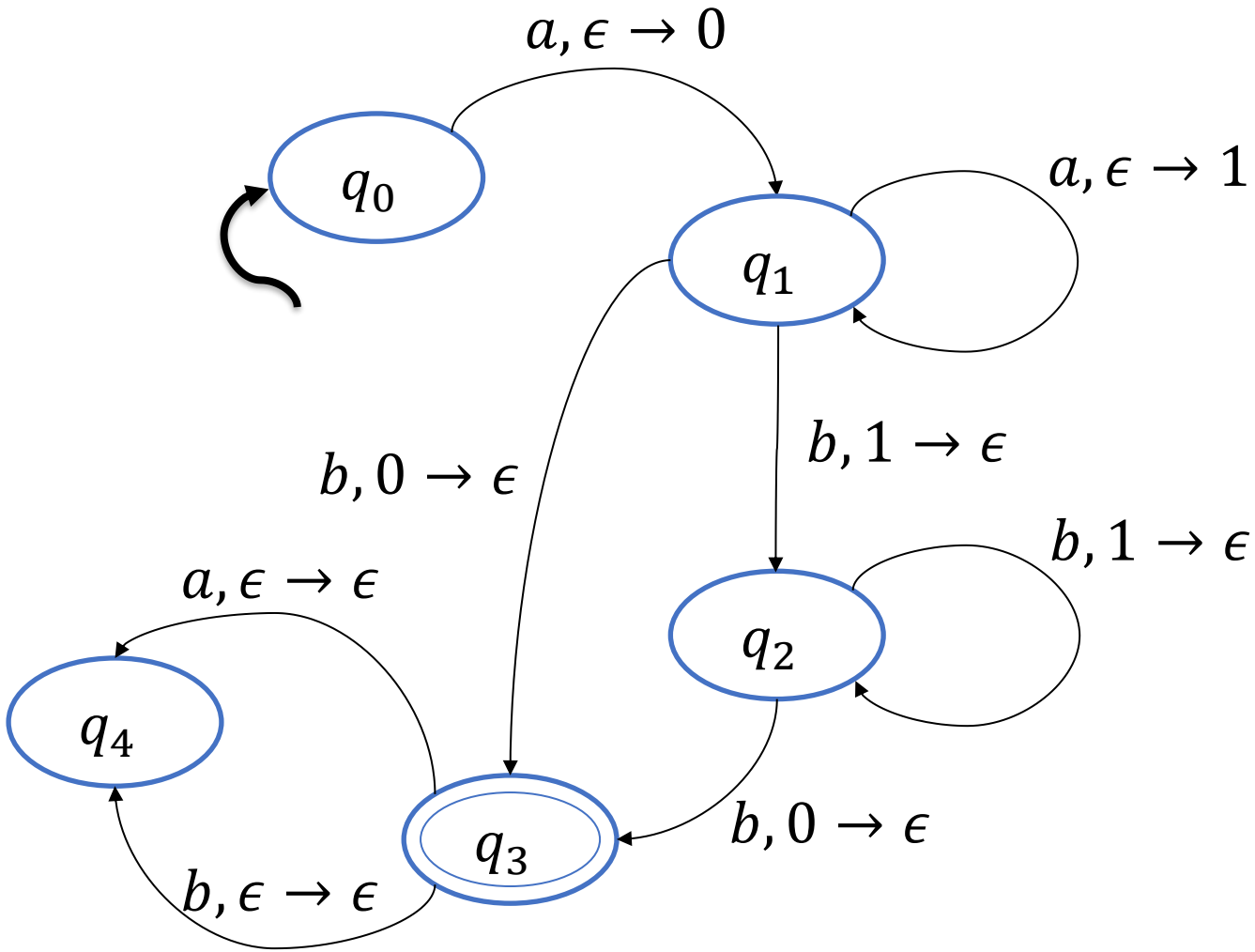
- Leggi x dall'input
- Leggi y dalla pila
- Metti z in memoria



Esecuzione di un Automa a Pila – Esempio

INPUT

a	a	b	b



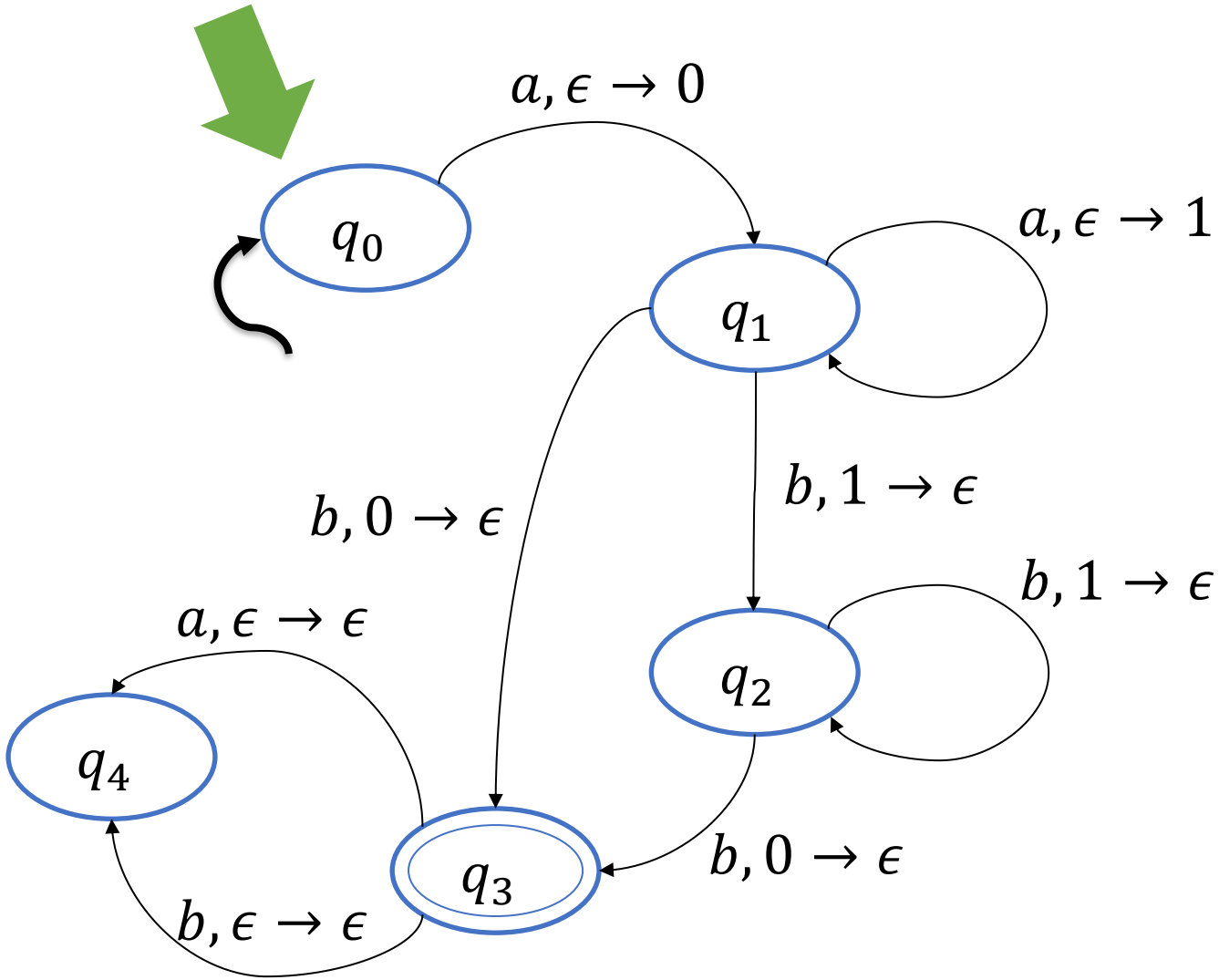
Esecuzione di un Automa a Pila – Esempio

MEMORIA CORRENTE

Memoria

INPUT

a	a	b	b



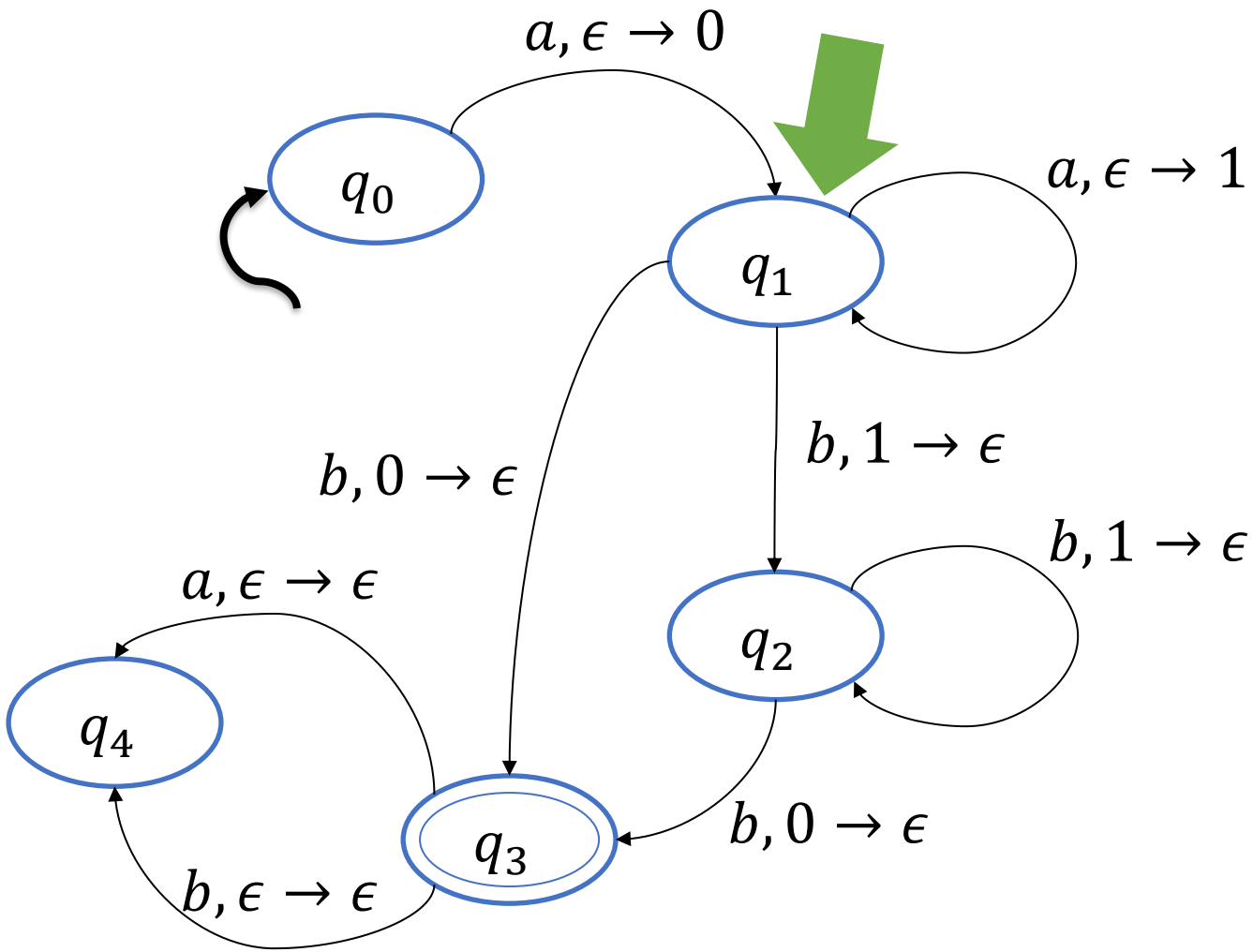
Esecuzione di un Automa a Pila – Esempio

MEMORIA CORRENTE

Memoria
0

INPUT

a	a	b	b



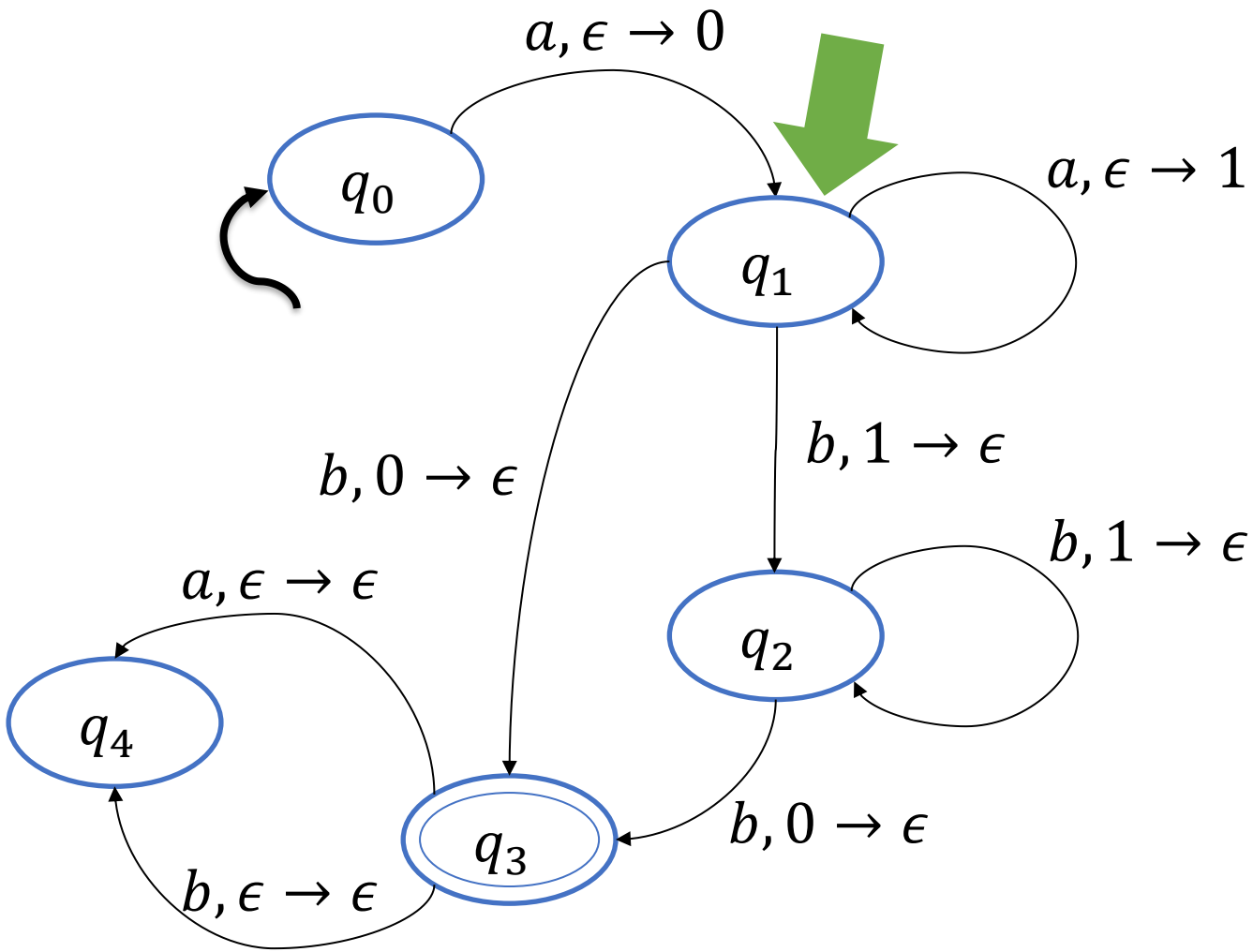
Esecuzione di un Automa a Pila – Esempio

MEMORIA CORRENTE

Memoria
0
1

INPUT

a	a	b	b



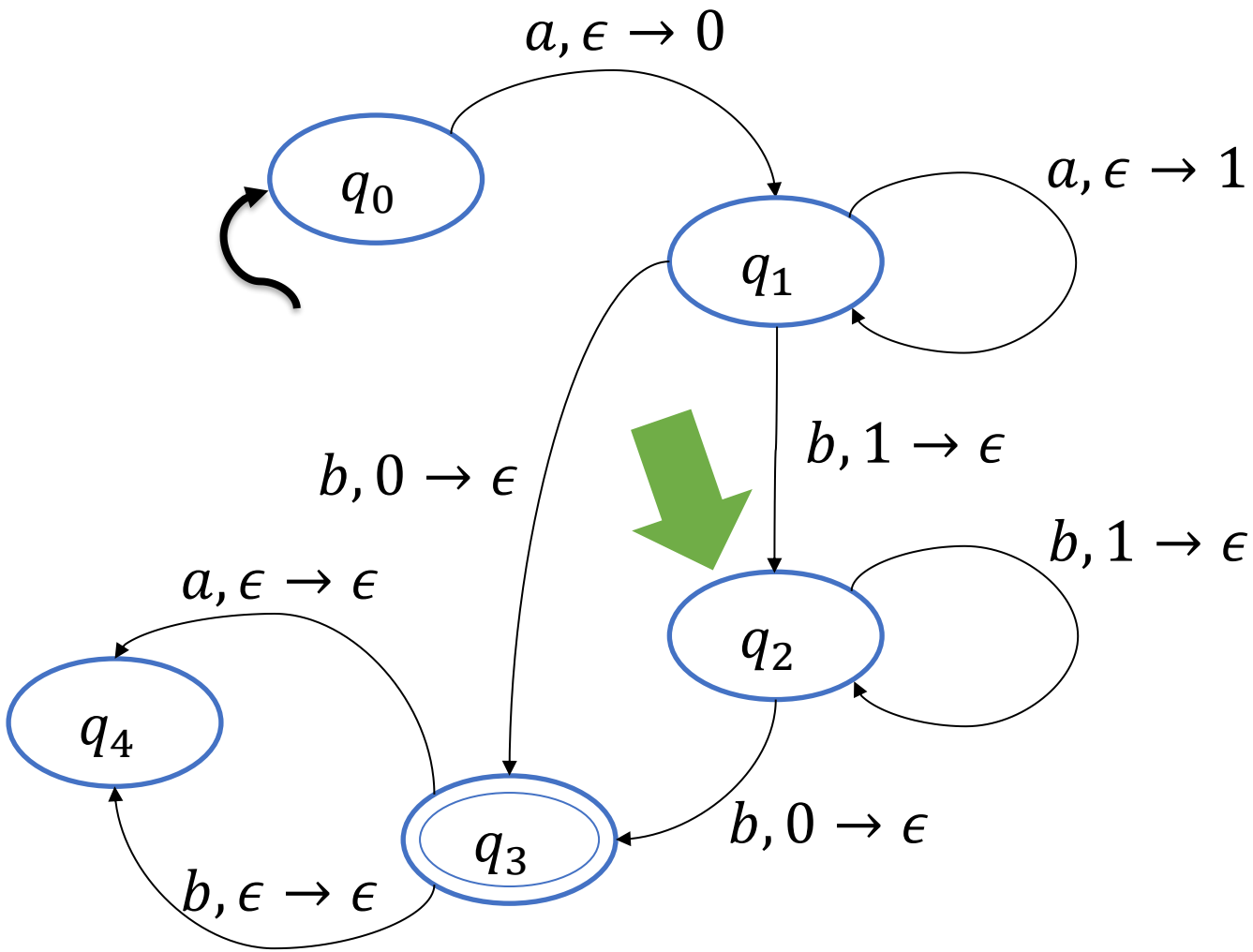
Esecuzione di un Automa a Pila – Esempio

MEMORIA CORRENTE

Memoria
0

INPUT

a	a	b	b



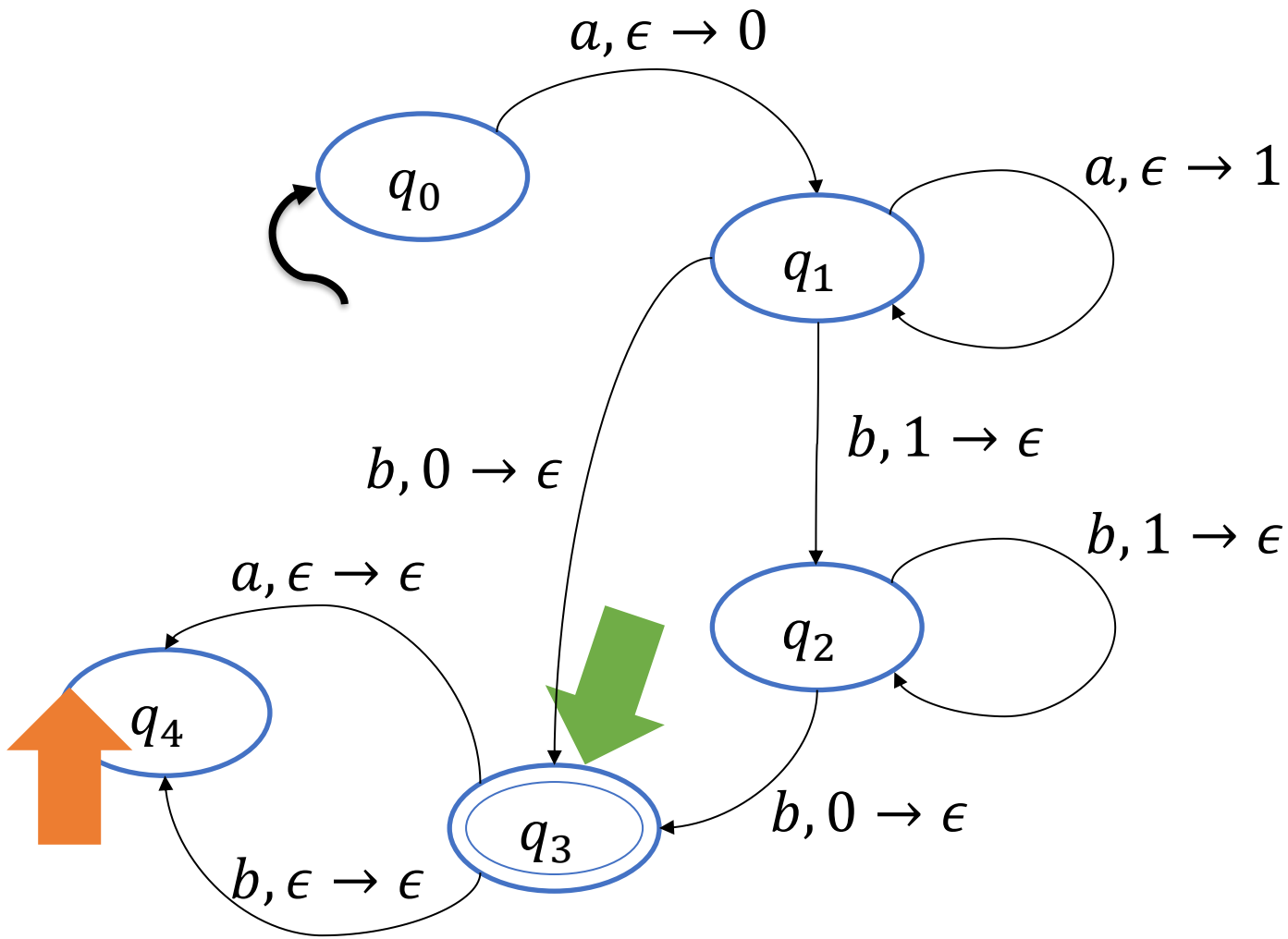
Esecuzione di un Automa a Pila – Esempio

MEMORIA CORRENTE

Memoria

INPUT

a	a	b	b



Esecuzione di un Automa a Pila – Esempio

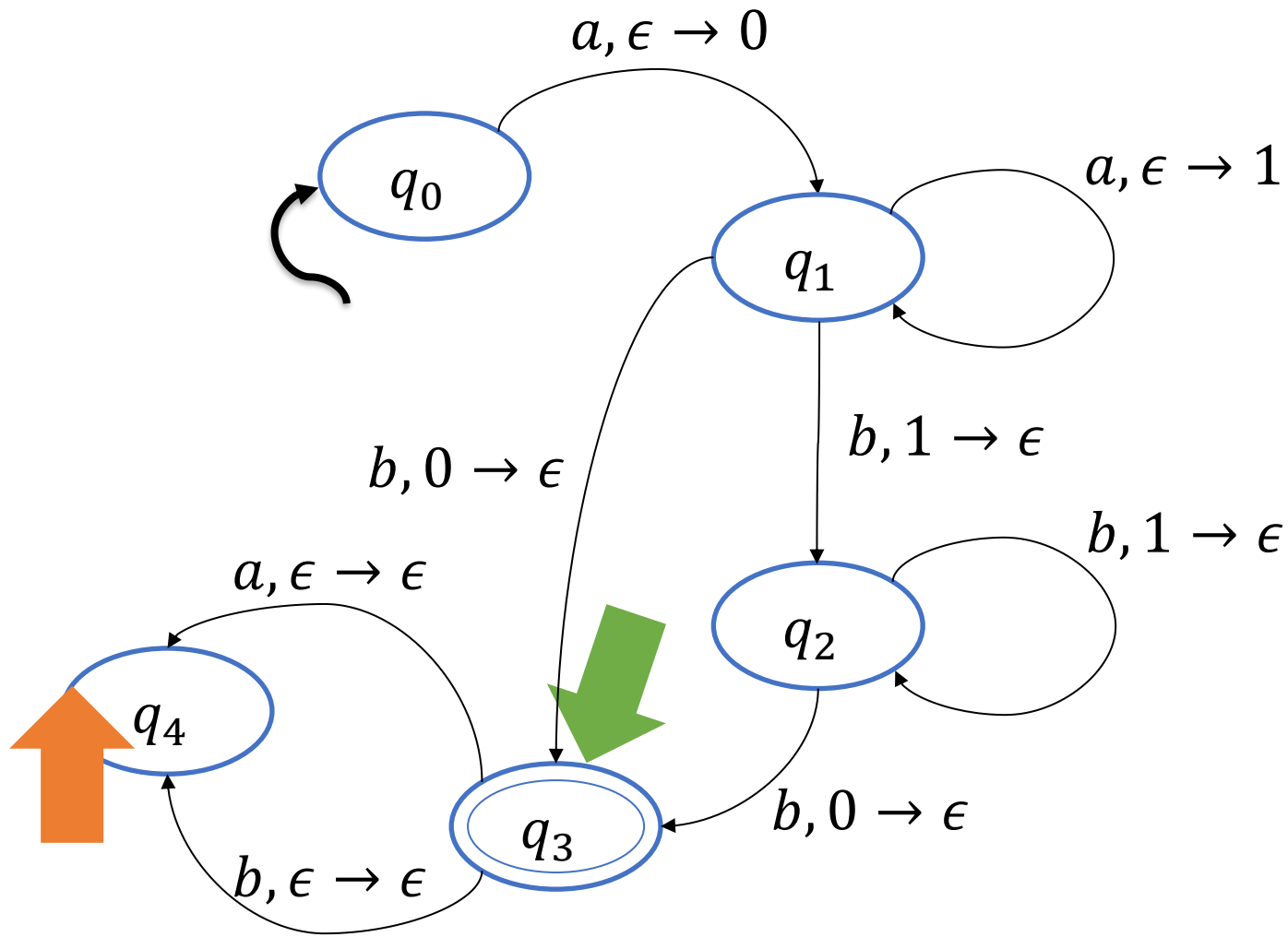
MEMORIA CORRENTE

Memoria

INPUT

a	a	b	b

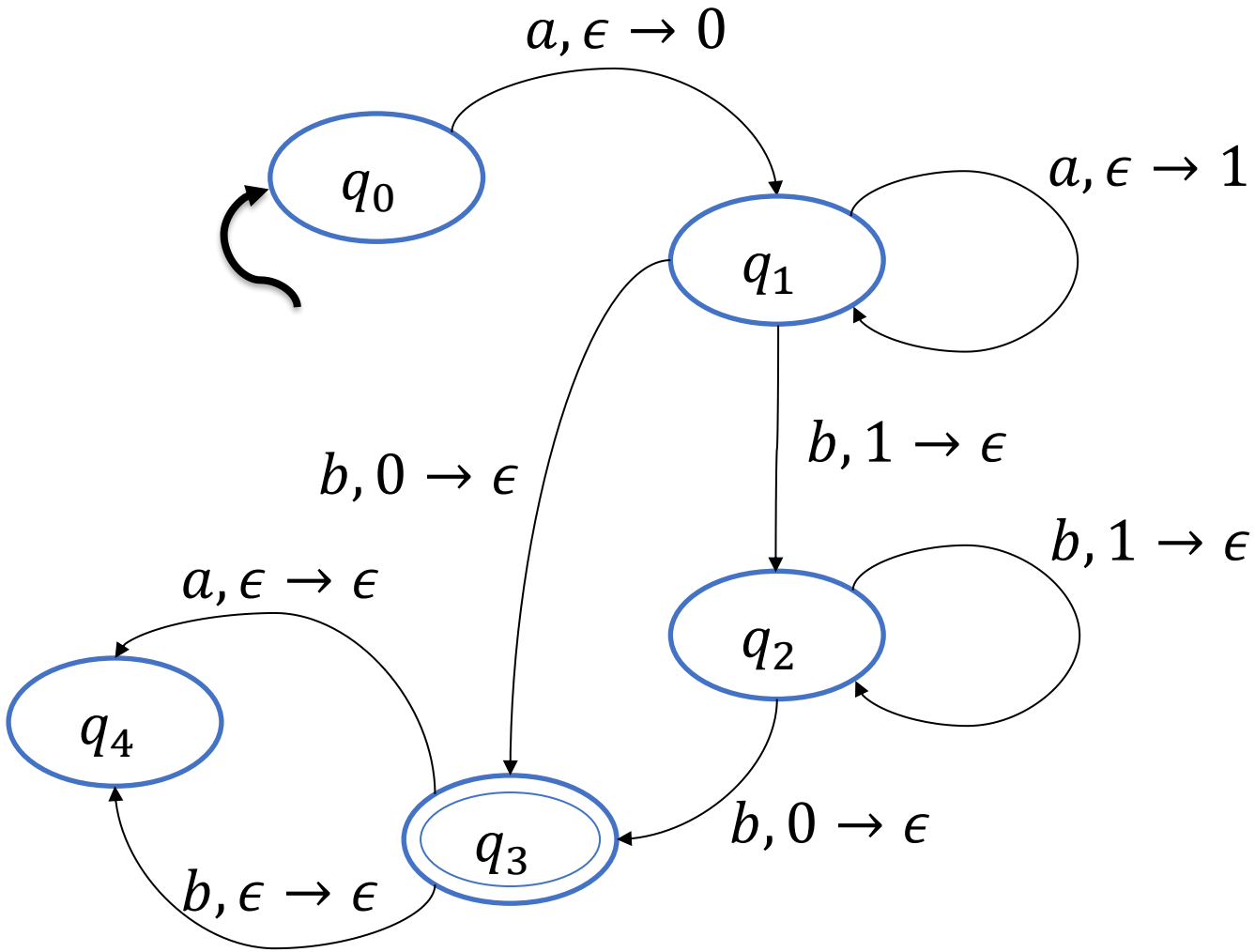
Esecuzione Accettante



Esecuzione di un Automa a Pila – Esempio

INPUT

a	a	a	b



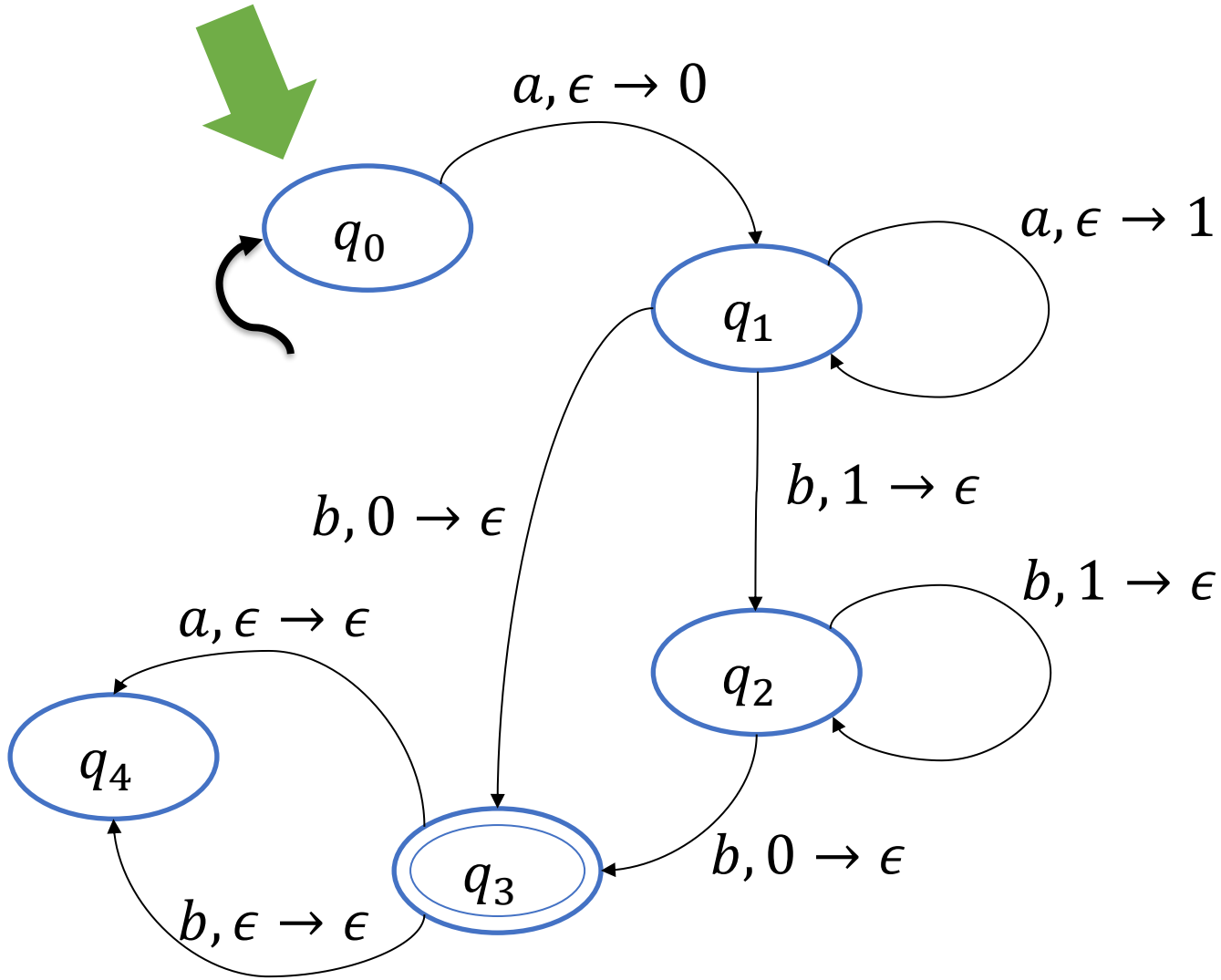
Esecuzione di un Automa a Pila – Esempio

MEMORIA CORRENTE

Memoria

INPUT

a	a	a	b



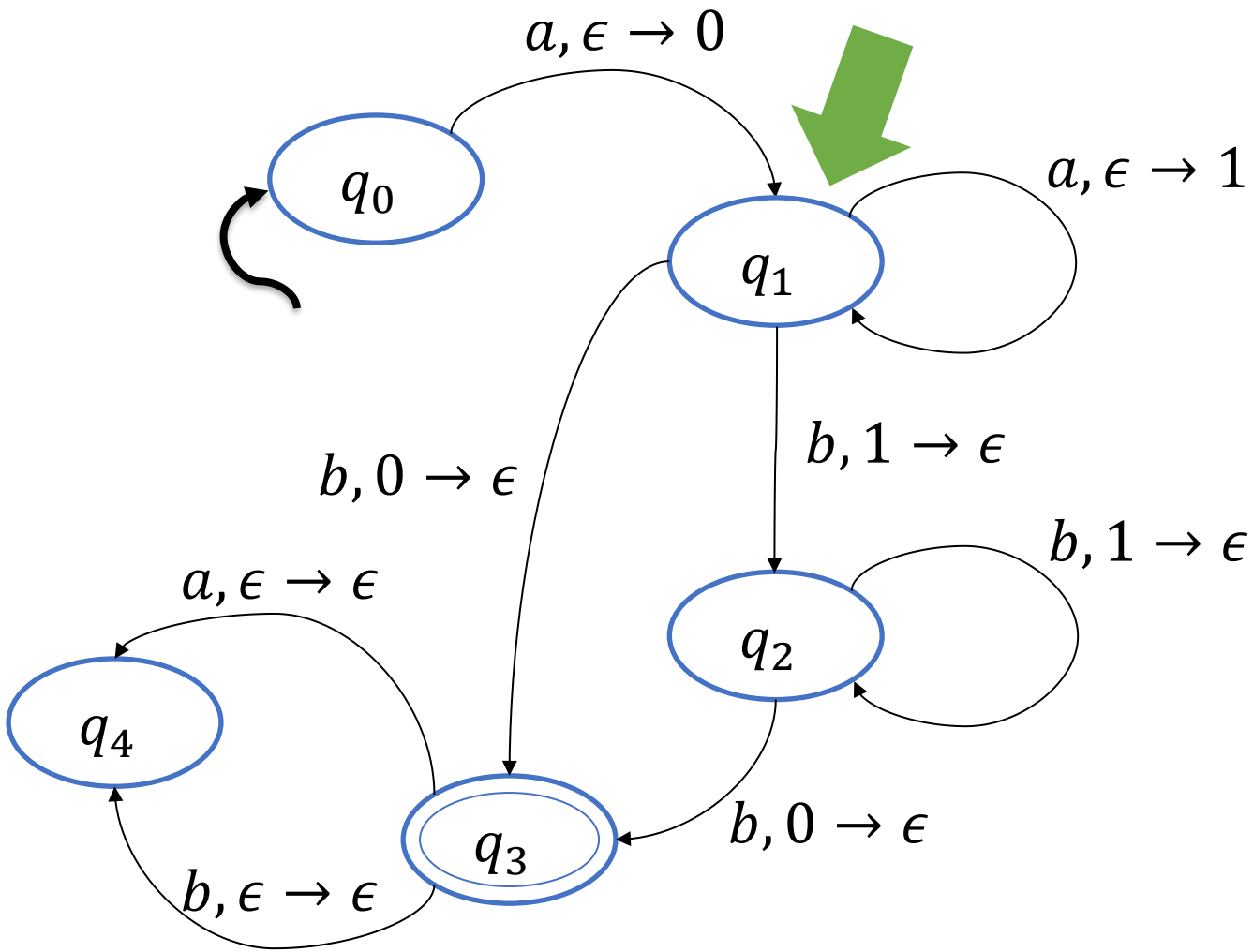
Esecuzione di un Automa a Pila – Esempio

MEMORIA CORRENTE

Memoria
0

INPUT

a	a	a	b



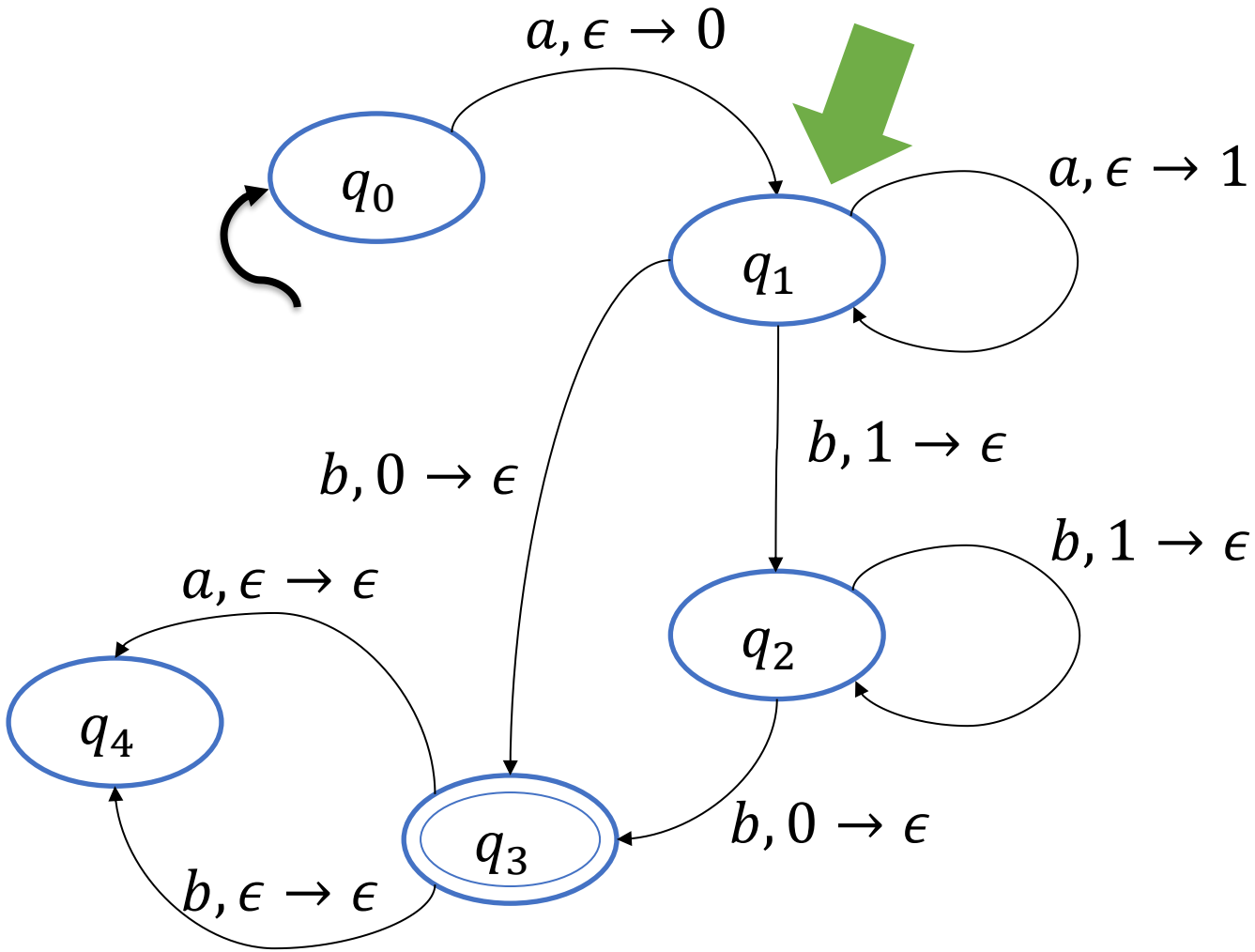
Esecuzione di un Automa a Pila – Esempio

MEMORIA CORRENTE

Memoria
0
1

INPUT

a	a	a	b



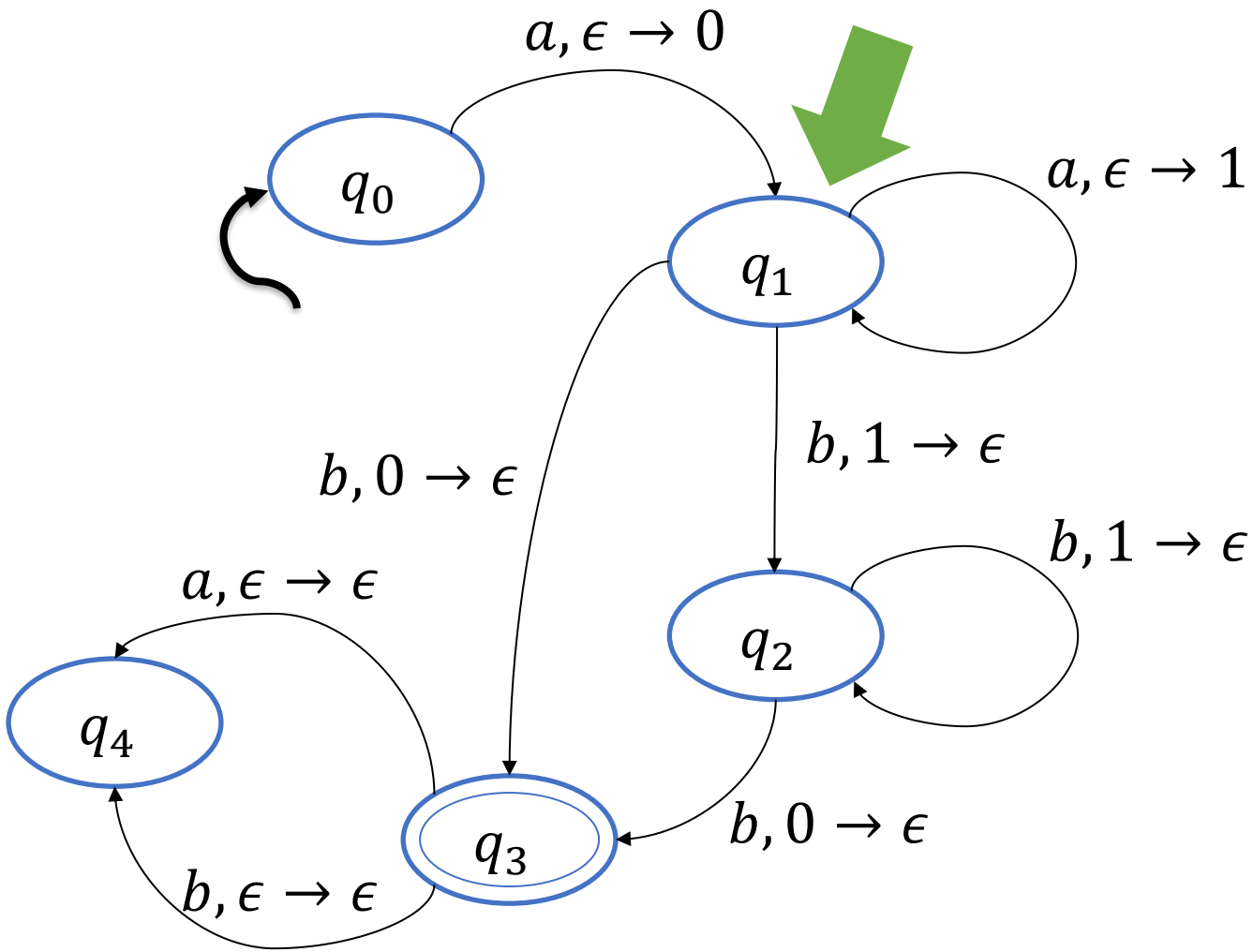
Esecuzione di un Automa a Pila – Esempio

MEMORIA CORRENTE

Memoria
0
1
1

INPUT

a	a	a	b



Esecuzione di un Automa a Pila – Esempio

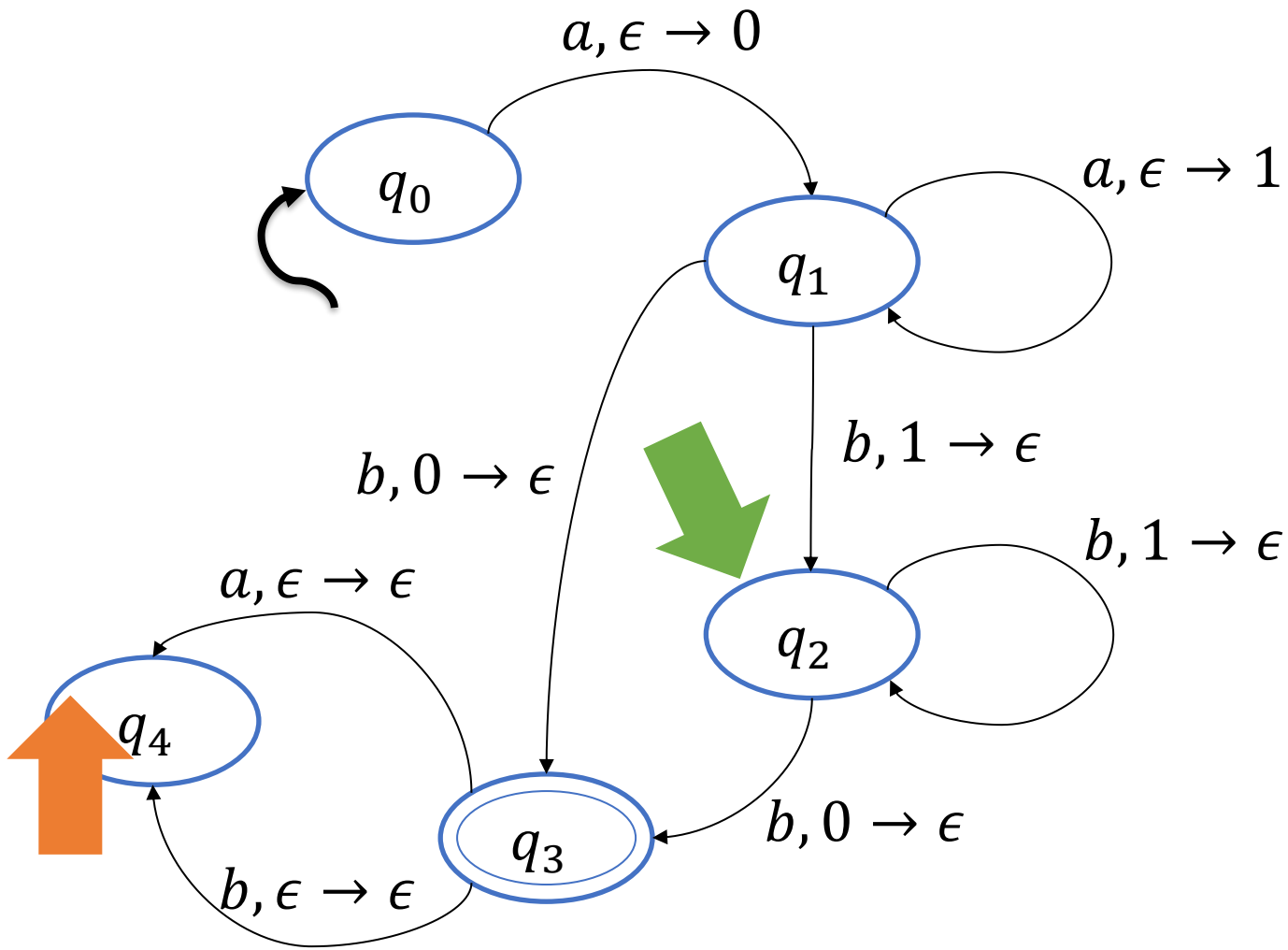
MEMORIA CORRENTE

Memoria
0
1

INPUT

a	a	a	b

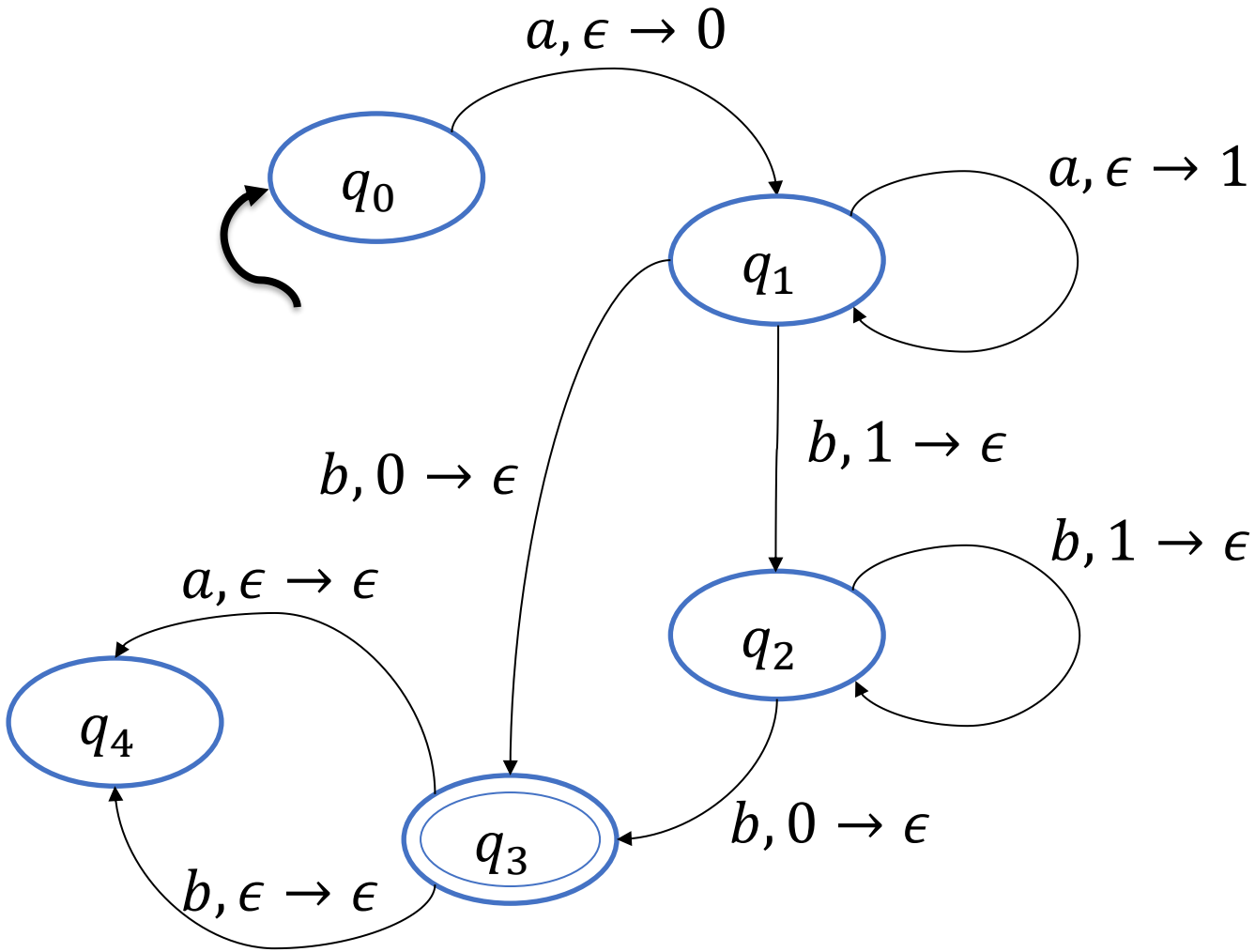
Esecuzione NON Accettante



Esecuzione di un Automa a Pila – Esempio

INPUT

a	b	b	b



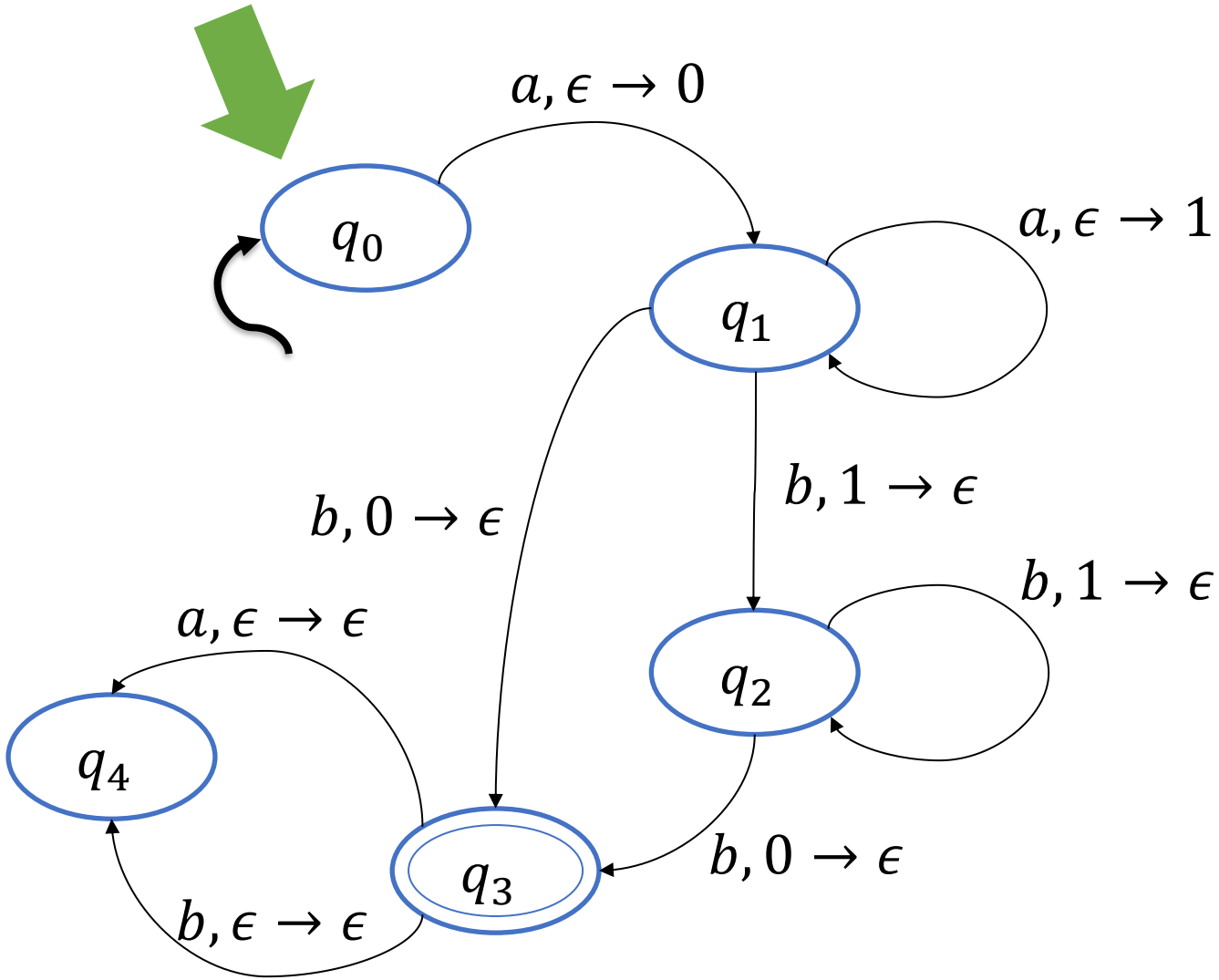
Esecuzione di un Automa a Pila – Esempio

MEMORIA CORRENTE

Memoria

INPUT

a	b	b	b



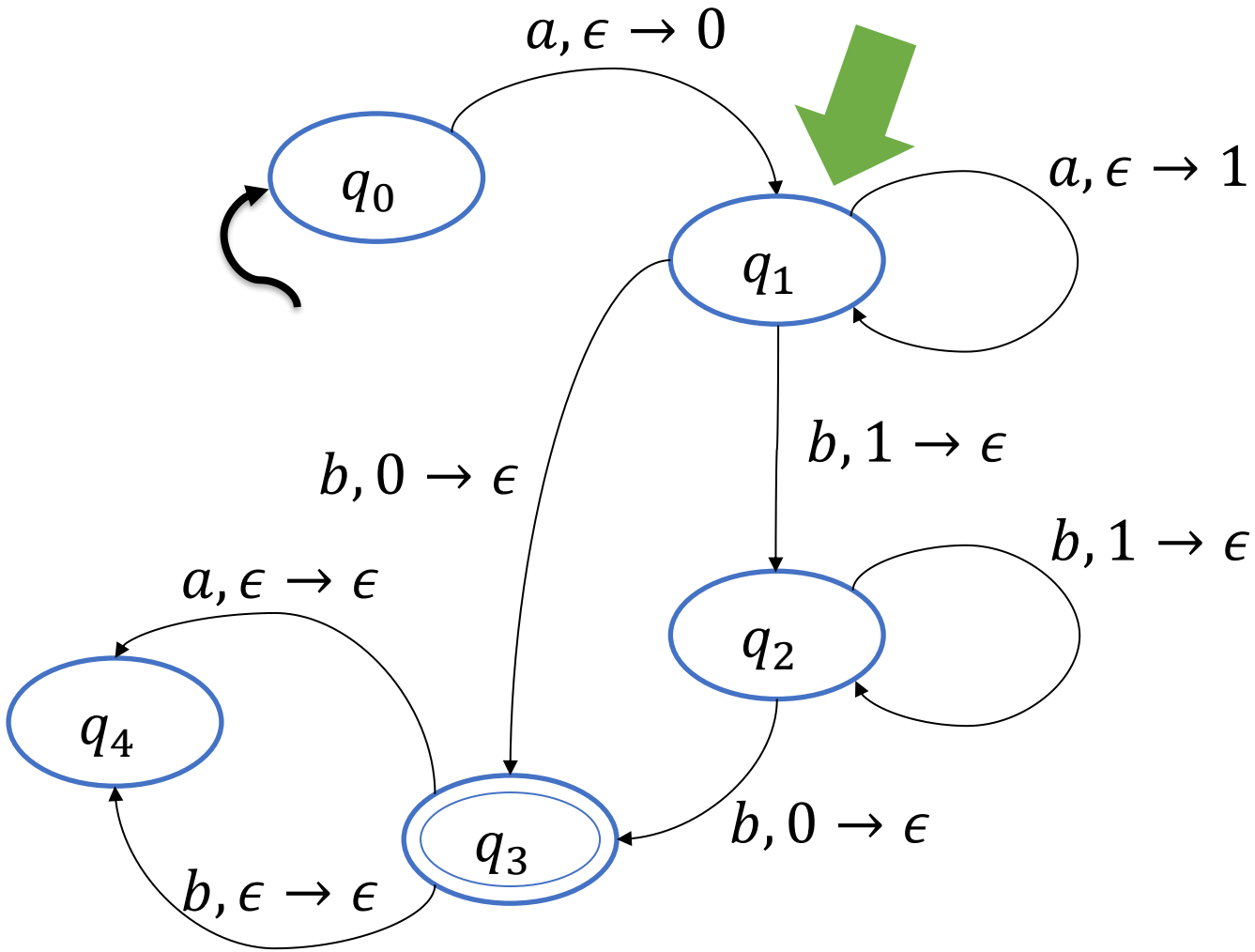
Esecuzione di un Automa a Pila – Esempio

MEMORIA CORRENTE

Memoria
0

INPUT

a	b	b	b



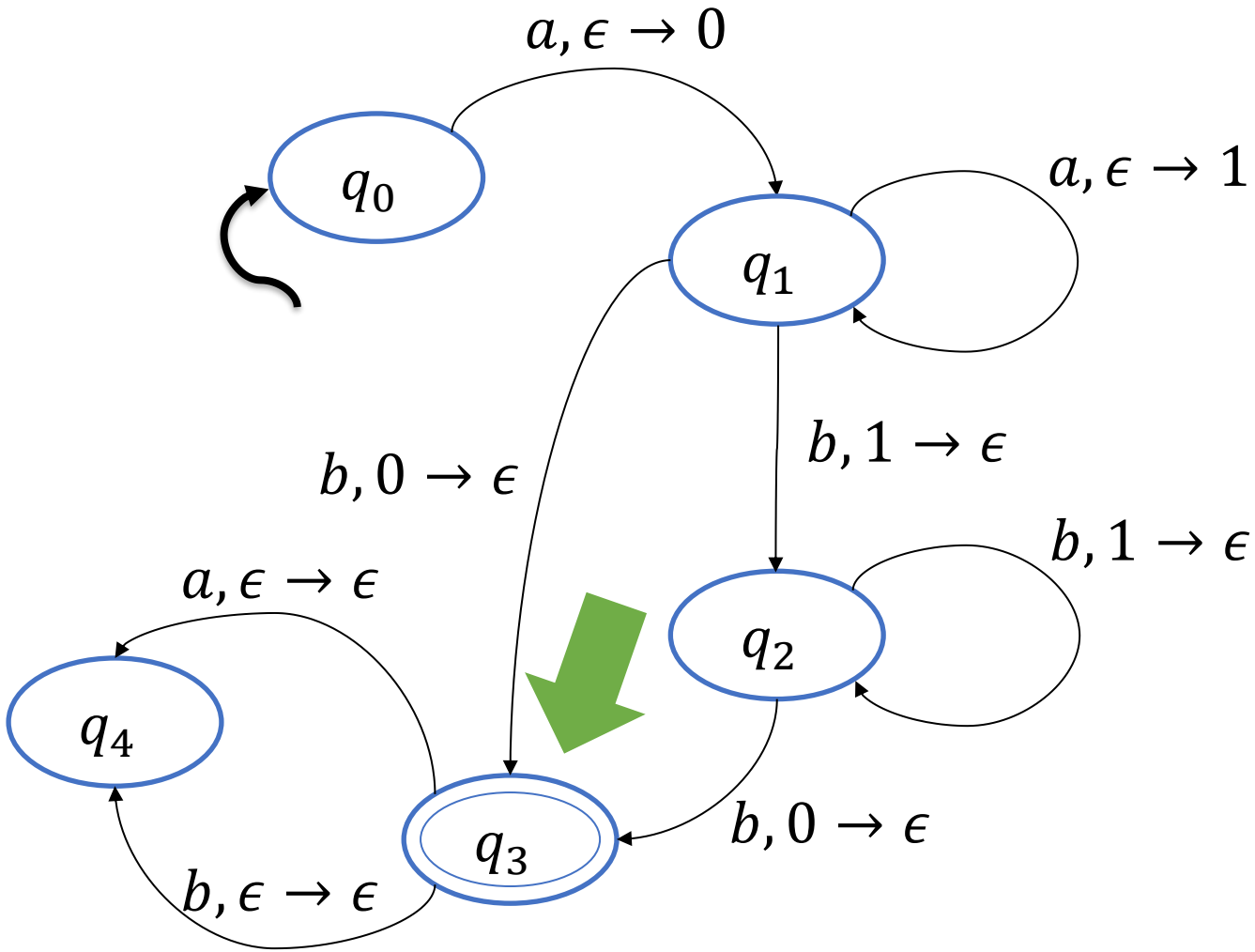
Esecuzione di un Automa a Pila – Esempio

MEMORIA CORRENTE

Memoria
0

INPUT

a	b	b	b



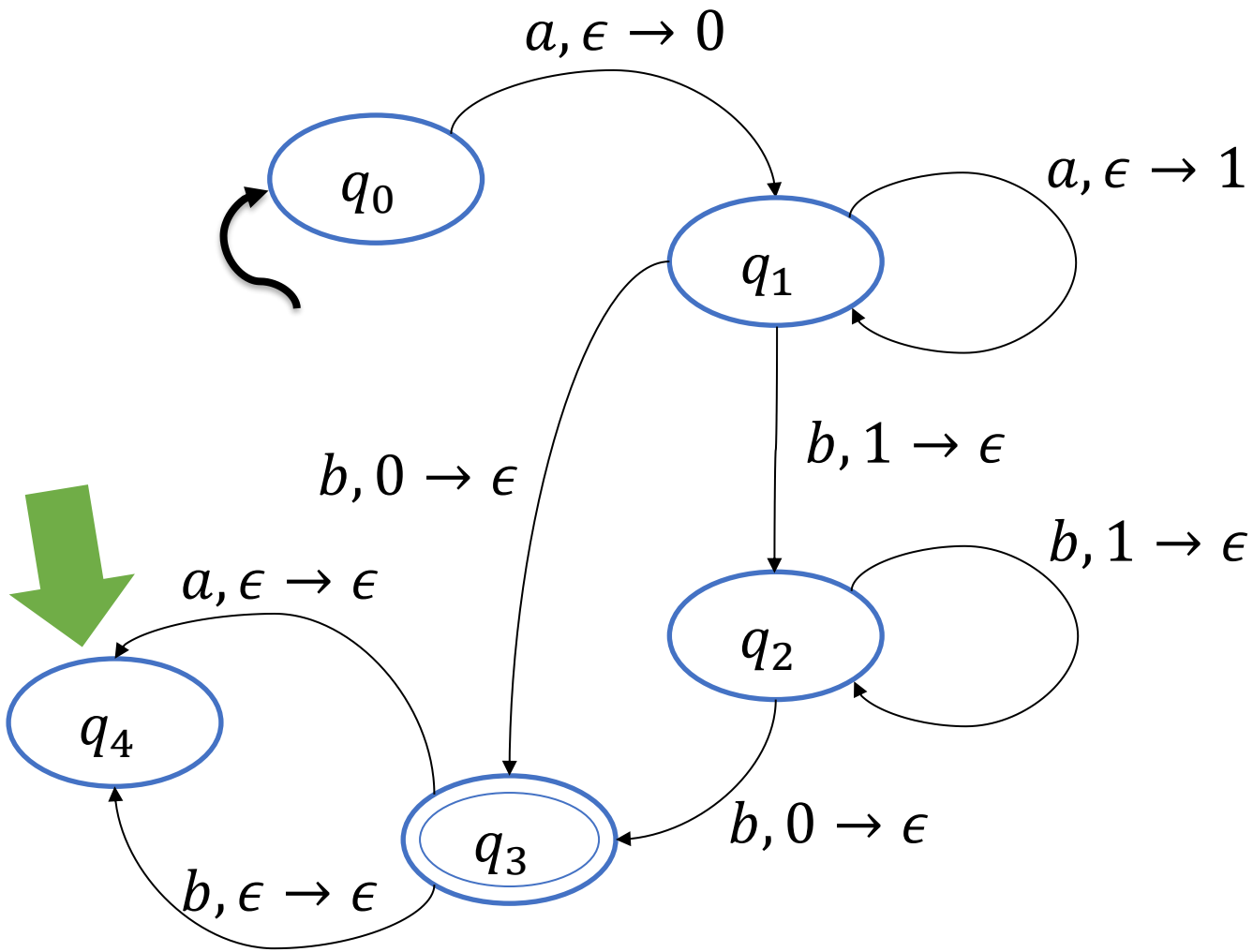
Esecuzione di un Automa a Pila – Esempio

MEMORIA CORRENTE

Memoria
0

INPUT

a	b	b	b



NON esiste una esecuzione accettante

Linguaggi Non Contestuali

Automa a Pila – Utilizzo Pratico

- Abbiamo definito gli Automi a Pila, un **modello computazionale** che estende gli ASF con una forma di limitata di memoria interna
  - **Memoria a Pila**

- Abbiamo mostrato un linguaggio che Automi a Pila riconosco e che non è un regolare
  - Ce ne sono moltissimi in realtà

- **Definizione**. Un **Linguaggio Non-Contestuale** è un linguaggio riconosciuto da un **Automa a Pila**

- **Proposizione**. Ogni **Linguaggio Regolare** è un **Linguaggio Non-Contestuale**
  - La **dimostrazione** è banale e consiste nell'osservare che ogni ASFD può essere implementato come un Automa a Pila (che non usa la Pila ☺)

- I **Linguaggi Non-Contestuale** sono molto usati per definire linguaggi di programmazione e linguaggi formali perché ammetto una rappresentazione semplice e compatta denominata **Grammatica Non-Contestuale**
  - Che non ha nulla a che fare con un Automa a Pila ☺

Grammatiche Non Contestuali

- **Definizione.** Una grammatica non contestuale (**context free**) è una 4-upla $\langle V, \Sigma, R, S \rangle$ dove
  - V è un insieme di simboli detti **non-terminali**
  - Σ con $\Sigma \cap V = \emptyset$ è un insieme di simboli detti **terminali**
  - R è un insieme finito di coppie (v, R) con $v \in V$ e $R \in (\Sigma \cup V)^*$ detto **insieme di regole**
    - R è una stringa sull'alfabeto $(\Sigma \cup V)$
  - $S \in V$ è detto **simbolo iniziale**

- **Esempio.** Intuitivamente, una grammatica non contestuale rappresenta tutte le stringhe di simboli terminali che si possono produrre a partire dal simbolo iniziale S applicando le regole in R

Grammatiche Non Contestuali

- **Esempio.** La seguente è una grammatica non contestuale
  - $V = \{formula\}$
  - $\Sigma = \{(,\ ),\ +,\ -,\ \times,\ \div,\ 0, 1\}$
  - $S = formula$
  - R contiene le seguenti regole
    - $\boldsymbol{formula} \rightarrow (\ \boldsymbol{formula}\ + \boldsymbol{formula}\ )$
    - $\boldsymbol{formula} \rightarrow (\ \boldsymbol{formula}\ - \boldsymbol{formula}\ )$
    - $\boldsymbol{formula} \rightarrow (\ \boldsymbol{formula}\ \times\ \boldsymbol{formula}\ )$
    - $\boldsymbol{formula} \rightarrow (\ \boldsymbol{formula} \div\ \boldsymbol{formula}\ )$
    - $\boldsymbol{formula} \rightarrow \mathbf{0}$
    - $\boldsymbol{formula} \rightarrow \mathbf{1}$

- Tale grammatica rappresenta il linguaggio delle espressioni algebriche che utilizzano i simboli $+, -, \times, \div, 0, 1$

Grammatiche Non Contestuali – Linguaggio Generato

- **Definizione 1.** Data una Grammatica Non Contestuale $G = \langle V, \Sigma, R, S \rangle$ diciamo che la stringa $c_1 \dots c_{k-1} c_k \; c_{k+1} \dots c_n \in (\Sigma \cup V)^*$ **genera la stringa** $c_1 \dots c_{k-1} s_1 \dots s_m \; c_{k+1} \dots c_n$ in G e esiste una regola $(c_k, s_1 \dots s_m)$ in R **e scriviamo**

$$c_1 \dots c_{k-1} c_k \; c_{k+1} \dots c_n \Rightarrow_G c_1 \dots c_{k-1} s_1 \dots s_m \; c_{k+1} \dots c_n$$

- **Definizione 2.** Data una Grammatica Non Contestuale $G = \langle V, \Sigma, R, S \rangle$ diciamo che la stringa $s_0 \in (\Sigma \cup V)^*$ **deriva la stringa** $s_n \in (\Sigma \cup V)^*$ in G se esiste una sequenza $s_0, s_1, l_2, \dots, s_n$ tale che $s_i \Rightarrow_G s_{i+1}$, per ogni $i = 0, n-1$ **e scriviamo**

$$s_0 \Rightarrow_G^* s_n$$

- **Definizione 3.** Data una Grammatica Non Contestuale $G = \langle V, \Sigma, R, S \rangle$, il linguaggio $L(G)$ riconosciuto da G è definito come $L(G) = \{w \mid S \Rightarrow_G^* w\}$
  - $L(G)$ è il linguaggio di tutte le stringhe che derivano dal simbolo iniziale S di G

Proprietà delle Grammatiche Non Contestuali

- Possiamo dimostrare una equivalenza fra le grammatiche contestuali e gli automi a pila

- **Teorema.** Un linguaggio S è **non contestuale** se e solo è esiste una grammatica non contestuale G tale che $S = L(G)$
- **Corollario.** Un linguaggio è riconosciuto da un automa a pila se e solo se è generato da una grammatica non contestuale

- Non vedremo la prova di questo teorema (è sul libro) ma ci dice che gli automi a pila possono essere utilizzati per riconoscere una **grandissima famiglia di linguaggi**
  - Praticamente tutti i linguaggi di programmazione moderni
  - Praticamente tutti i linguaggi logici

Grammatiche Non Contestuali – Esempio

- La grammatica non contestuale G seguente genera le **formule della logica proposizionale**
  - Non-Terminali : $\{formula, variabile\}$,
  - Terminali $\Sigma = \{(,\ ),\ \wedge,\ \vee,\ \neg,\ V_1, V_2, \dots, V_n\}$
  - Simbolo Iniziale $formula$
  - G contiene le seguenti regole
    - $formula \rightarrow variabile$
    - $formula \rightarrow (\ formula \wedge formula\ )$
    - $formula \rightarrow (\ formula \vee formula\ )$
    - $formula \rightarrow \neg(formula\ )$
    - $formula \rightarrow (\ formula \div\ formula\ )$
    - $variabile \rightarrow V_1$
    - $variabile \rightarrow V_2$
    - $\dots$
    - $variabile \rightarrow V_n$

Grammatiche Non Contestuali – Esempio

- La grammatica non contestuale G precedente ci fa concludere quanto segue

- **Proposizione 1.** Il linguaggio delle formule della logica proposizionale è un linguaggio contestuale
  – Vedi la grammatica precedente

- **Corollario.** Esiste un Automa a Pila M tale che $\mathrm{L}(M)$ è il linguaggio delle formule proposizionali
  – A causa dell'equivalenza che abbiamo definito in precedenza.

- **Domanda.** Esiste un linguaggio $\mathcal{L}$ che non è non contestuale?
  - In altre parole, esiste un linguaggio che non può essere riconosciuto da un automa a pila
  - **Ovvero** che non può essere generato da una grammatica non contestuale

Pumping Lemma Per Linguaggi Non Contestuali

- Anche per i linguaggi non contestuali possiamo dimostrare una forma di Pumping Lemma
  - Stringhe lunghe esibiscono una struttura ricorrente che ci permette di fare Pumping e ottenere altre stringhe nel linguaggio

- **Lemma. [Pumping lemma per linguaggi non contestuali]** Per ogni linguaggio non contestuale $\mathcal{L}$ di cardinalità infinita esiste una costante n tale che: se $z \in L$ e $|z| \geq n$, allora possiamo scrivere $z = uvwxy$, con $|vx| \geq 1$ e $|vwx| \leq n$, e ottenere che $uv^i wx^i y \in L$ per ogni $i \geq 0$

- **Non vediamo i dettagli della prova (è sul libro).** L'intuizione è simile a quella del Pumping Lemma per I linguaggi regolari: il numero di simboli non terminali nella grammtica che genera il linguaggio ci permette di concludere la una struttura ricorrente

Linguaggi NON Non Contestuali

- **Proposizione**. Il linguaggio $L = \{a^m b^m c^m | \, m \geq 1\}$ non è non contestuale

- Dobbiamo dimostrare che: per ogni costante n, esiste una stringa $z \in L$ con $|z| \geq n$ tale che, per ogni possible suddivisione $uvwxy$ di z con $|vx| \geq 1$ e $|vwx| \leq n$, abbiamo che esiste un $i \geq 0$ tale per cui $uv^i wx^i y \notin L$

- **Dimostrazione** : Fissiamo una qualunque stringa z tale che $z \in L$ e $|z| = n$. Considera ogni possibile suddivisione $uvwxy$ di z con $|vx| \geq 1$ e $|vwx| \leq n$. Abbiamo due casi possibili:
  1. Uno tra v ed x contiene almeno due simboli diversi. In questo caso, si deriva che la stringa uv^2wx^2y non appartiene ad L perché rompe l'ordine dei simboli
  2. Se v ed x contengono un solo simbolo, allora la stringa uv^2wx^2y non appartiene ad L perché la cardinalità delle sotto-stringhe di a, b e c è diversa