COM3064 Automata Theory

Week 7: Chomsky Normal Form & Pushdown Automata

Lecturer: Dr. Sevgi YİĞİT SERT
Spring 2023

Resources: Introduction to The Theory of Computation, M. Sipser,
Introduction to Automata Theory, Languages, and Computation, J.E. Hopcroft, R. Motwani, and J.D. Ullman
BBM401 Automata Theory and Formal Languages, İlyas Çiçekli
CENG280 Formal Languages and Abstract Machines, Halit Oğuztüzün

Chomsky Normal Form

A context-free grammar is in *Chomsky Normal Form* if every rule is of the form

$$A \rightarrow BC$$

$$A \rightarrow a$$

where a is any terminal and A, B, and C are any variables — except that B and C may not be the start variable.

In addition, we permit the rule $S \rightarrow \varepsilon$ where S is the start variable if the language of the grammar contains ε.

Chomsky Normal Form

Theorem: Every (non-empty) context free language can be generated by a context-free grammar in Chomsky normal form.

- In order to obtain **an equivalent grammar in Chomsky normal form** for any given CFG G, we will have the following conversion steps:
  1. **Add a new start variable S_0** and a new production rule $S_0 \rightarrow S$ where S is the original start variable of G.
  2. **Eliminate ε-productions** (productions of the form $A \rightarrow \varepsilon$). After this conversion step, only one ε-production ($S_0 \rightarrow \varepsilon$) if the language of G contains ε.
  3. **Eliminate unit productions** (productions of the form $A \rightarrow B$ where A and B are variables).
  4. **Eliminate useless symbols**. Useless symbols do not appear in any derivation of a terminal string from the start symbol.
  5. **Convert the remaining rules into Chomsky normal form** adding new variables and rules.

Add A New Start Variable

- We **add a new start variable S_0** and the **rule $S_0 \to S$**, where S was the original start variable.
- This change guarantees that the start variable does not occur on the right-hand side of a rule.
- The new grammar is equivalent to the original grammar (i.e ***they generate same language***).

Example:

$$S \to \varepsilon \mid 0S1$$

$$\Rightarrow$$

$$\mathbf{S_0 \to S}$$
$$S \to \varepsilon \mid 0S1$$

equivalent grammars

Eliminate ε-productions

- In order to remove ε-**productions**, first we will determine **nullable variables.**
- A variable A is said to **nullable** if $\mathbf{A} \overset{*}{\Rightarrow} \boldsymbol{\varepsilon}$

A CFG G_1

$S_0 \rightarrow S$

$S \rightarrow \varepsilon \mid 0S1$

➔ nullable(G_1) = $\{S, S_0\}$

A CFG G_2

$S_0 \rightarrow S$

$S \rightarrow AB$

$A \rightarrow aAA \mid \varepsilon$

$B \rightarrow bBB \mid \varepsilon$

➔ nullable(G_2) = $\{A, B, S, S_0\}$

Eliminate ε-productions

Steps for ε-production elimination for CFG G:

1. **Find nullable(G),** the set of all *nullable* symbols of G.
2. **Generate new rules** from a rule R by eliminating *nullable* variables from its right-side, if *nullable* variables appears on its right-side.
   - The number of new rules depends on the number of *nullable* variables on the right-side. If there are k *nullable* variables, we have to generate 2^k-1 new rules.
   - ***Generated new rule is added if it is not already among the rules***.
     - **If $R \to \alpha A\beta$ is a rule and A is the only *nullable* variable on $\alpha A\beta$, generate and add the new rule $R \to \alpha\beta$.**
     - **If $R \to \alpha A\beta B\gamma$ is a rule and A and B are only *nullable* variables on $\alpha A\beta B\gamma$, generate and add the new rules $R \to \alpha\beta B\gamma$, $R \to \alpha A\beta\gamma$ and $R \to \alpha\beta\gamma$.**

3. **Remove all ε-productions** $A \to \varepsilon$ (except $S_0 \to \varepsilon$) from the rules.

The new grammar that is obtained by **eliminating ε-productions** is equivalent to the original grammar (i.e ***they generate same language***).

Eliminate ε-productions - Example

$S_0 \rightarrow S$ $\quad\quad$ $S \rightarrow \varepsilon \mid 0S1$

- nullable(G) = $\{S, S_0\}$
- Since S is nullable,
  - generate $S_0 \rightarrow \varepsilon$ from $S_0 \rightarrow S$
- Since S is nullable,
  - generate $S \rightarrow 01$ from $S \rightarrow 0S1$
- After all generations, we have the following rules:

  $S_0 \rightarrow \varepsilon \mid S$

  $S \rightarrow \varepsilon \mid 01 \mid 0S1$

- Remove all ε-productions except $S_0 \rightarrow \varepsilon$, our final grammar is:

  $S_0 \rightarrow \varepsilon \mid S$

  $S \rightarrow 01 \mid 0S1$

equivalent grammars

Eliminate ε-productions - Example

$\mathbf{S_0 \rightarrow S} \qquad \mathbf{S \rightarrow AB} \qquad \mathbf{A \rightarrow aAA} \mid \cancel{\varepsilon} \qquad \mathbf{B \rightarrow bBB} \mid \cancel{\varepsilon}$

- nullable(G) = {A, B, S, S_0}
- Generate new rules:

$S_0 \rightarrow \varepsilon$			from $S_0 \rightarrow S$
$S \rightarrow A$	$S \rightarrow B$	$S \cancel{\rightarrow} \varepsilon$	from $S \rightarrow AB$
$A \rightarrow aA$	$A \cancel{\rightarrow} aA$	$A \rightarrow a$	from $A \rightarrow aAA$
$B \rightarrow bB$	$B \cancel{\rightarrow} bB$	$B \rightarrow b$	from $B \rightarrow bBB$

- Remove ε-productions

$\mathbf{S_0 \rightarrow S} \mid \varepsilon$

$\mathbf{S \rightarrow AB \mid A \mid B}$

$\mathbf{A \rightarrow aAA \mid aA \mid a}$

$\mathbf{B \rightarrow bBB \mid bB \mid b}$

equivalent grammars

Eliminate Unit Productions

- $A \rightarrow B$ **is a unit production**, whenever A and B are variables.
- Unit productions can be eliminated from a grammar to obtain a grammar without unit productions.
  - The resulting grammar that is obtained by eliminating unit productions will be equivalent to the original grammar.

- We will remove unit productions one by one from the grammar.
- **Remove a unit production $A \rightarrow B$ from the grammar.**
  - **Then, whenever a rule $B \rightarrow u$ appears, we add the rule $A \rightarrow u$ unless this was a unit rule previously removed.**
- **We repeat these steps until we eliminate all unit rules.**

Eliminate Unit Productions - Example

$S_0 \rightarrow \varepsilon \mid \mathbf{S}$

$\mathbf{S} \rightarrow \mathbf{01} \mid \mathbf{0S1}$

- Unit productions: { $S_0 \rightarrow S$ }
- Remove $S_0 \rightarrow S$,
  - Add $S_0 \rightarrow 01$ and $S_0 \rightarrow 0S1$
- The resulting grammar after eliminating unit productions.

$S_0 \rightarrow \varepsilon \mid \mathbf{01} \mid \mathbf{0S1}$

$\mathbf{S} \rightarrow \mathbf{01} \mid \mathbf{0S1}$

equivalent grammars

Eliminate Unit Productions - Example

$S_0 \rightarrow S \mid \varepsilon$
$\mathbf{S \rightarrow AB \mid A \mid B}$
$\mathbf{A \rightarrow aAA \mid aA \mid a}$
$\mathbf{B \rightarrow bBB \mid bB \mid b}$

- Unit productions: { $\mathbf{S_0 \rightarrow S, S \rightarrow A, S \rightarrow B}$ }
- Remove $\mathbf{S \rightarrow B}$, add $\mathbf{S \rightarrow bBB \mid bB \mid b}$
- Remove $\mathbf{S \rightarrow A}$, add $\mathbf{S \rightarrow aAA \mid aA \mid a}$
- Remove $\mathbf{S_0 \rightarrow S}$, add $\mathbf{S_0 \rightarrow AB \mid aAA \mid aA \mid a \mid bBB \mid bB \mid b}$

• The resulting grammar after eliminating unit productions.

$\mathbf{S_0 \rightarrow \varepsilon \mid AB \mid aAA \mid aA \mid a \mid bBB \mid bB \mid b}$
$\mathbf{S \rightarrow AB \mid aAA \mid aA \mid a \mid bBB \mid bB \mid b}$
$\mathbf{A \rightarrow aAA \mid aA \mid a}$
$\mathbf{B \rightarrow bBB \mid bB \mid b}$

equivalent grammars

Eliminate Unit Productions - Example

$E \rightarrow E+T \mid T$
$T \rightarrow T*F \mid F$
$F \rightarrow G\text{^}F \mid G$
$G \rightarrow id \mid (E)$

- Unit productions: { $E \rightarrow T$, $T \rightarrow F$, $F \rightarrow G$ }
- Remove $F \rightarrow G$, add $F \rightarrow id \mid (E)$
- Remove $T \rightarrow F$, add $T \rightarrow G\text{^}F \mid id \mid (E)$
- Remove $E \rightarrow T$, add $E \rightarrow T*F \mid G\text{^}F \mid id \mid (E)$

• The resulting grammar after eliminating unit productions.

$E \rightarrow E+T \mid T*F \mid G\text{^}F \mid id \mid (E)$
$T \rightarrow T*F \mid G\text{^}F \mid id \mid (E)$
$F \rightarrow G\text{^}F \mid id \mid (E)$
$G \rightarrow id \mid (E)$

equivalent grammars

Eliminate Unit Productions - Example

Eliminating unit productions in different order do not change the result.

$E \rightarrow E+T \mid T$
$T \rightarrow T*F \mid F$
$F \rightarrow G\text{^}F \mid G$
$G \rightarrow id \mid (E)$

- Unit productions: { $E \rightarrow T$, $T \rightarrow F$, $F \rightarrow G$ }
- Remove $E \rightarrow T$, add $E \rightarrow T*F \mid F$
- Remove $T \rightarrow F$, add $T \rightarrow G\text{^}F \mid G$
- Remove $F \rightarrow G$, add $F \rightarrow id \mid (E)$
- Remove newly introduced unit productions
- Remove $E \rightarrow F$, add $E \rightarrow G\text{^}F \mid id \mid (E)$
- Remove $T \rightarrow G$, add $T \rightarrow id \mid (E)$

- The resulting grammar after eliminating unit productions.

$E \rightarrow E+T \mid T*F \mid G\text{^}F \mid id \mid (E)$
$T \rightarrow T*F \mid G\text{^}F \mid id \mid (E)$
$F \rightarrow G\text{^}F \mid id \mid (E)$
$G \rightarrow id \mid (E)$

equivalent grammars

Eliminate Useless Symbols

- A symbol X is **useful** for a grammar $G = (V, T, P, S)$, if there is a derivation

$$S \stackrel{*}{\Rightarrow} \alpha\, X\, \beta \stackrel{*}{\Rightarrow} w$$

  for a terminal string w.

- Symbols that **are not useful** are called **useless**.

- A symbol X is **generating** if $X \stackrel{*}{\Rightarrow} w$ for some string $w \in T^*$.
- A symbol X is **reachable** if $S \stackrel{*}{\Rightarrow} \alpha\, X\, \beta$ for some $\{\alpha, \beta\} \subseteq (V \cup T)^*$.

- If we eliminate **non-generating symbols** first, and then **non-reachable symbols**, we will **be left with only useful symbols**.
  - The grammar that is obtained by eliminating useless symbols will be equivalent to the original grammar.

Eliminate Useless Symbols - *computing generating symbols*

- For a grammar $G = (V, T, P, S)$, the generating symbols **generating(G)** are computed by the following closure algorithm:

Basis: **generating(G) = T**

Induction:

If **$X \to \varepsilon \in P$ or $X \to A_1 \ldots A_n \in P$ where $\{A_1, \ldots, A_n\} \subseteq$ generating(G)** then **generating(G) = generating(G) $\cup \{X\}$**

Example:

- Let G be $S \to AB \mid a, \quad A \to b$
- Initially, generating(G) = $\{a, b\}$
- A will be in generating(G) because of $A \to b$
- S will be in generating(G) because of $S \to a$

- Thus, generating(G) $= \{a, b, A, S\}$ and non-generating symbols are $\{B\}$

Eliminate Useless Symbols - *computing reachable symbols*

- For a grammar $G = (V, T, P, S)$, the reachable symbols **reachable(G)** are computed by the following closure algorithm:

Basis: **reachable(G) = {S}**

Induction:

If **X ∈ reachable(G) and X→ α ∈ P** then **add all symbols in α to reachable(G).**

Example:

- Let G be $S \to AB \mid a,\ A \to b, C \to a$
- Initially, reachable(G) = $\{S\}$
- A and B will be in reachable(G) because of $S \to AB$
- a will be in reachable(G) because of $S \to a$
- b will be in reachable(G) because of $A \to b$

- Thus, reachable(G)=$\{S, A, B, a, b\}$ and non-reachable symbols are $\{C\}$

Eliminate Useless Symbols

Steps to eliminate useless symbols from $\boldsymbol{G = (V, T, P, S)}$:

1. Compute generating(G).
2. Remove all productions containing at least one non-generating symbol in order to create a new grammar G_1 (a grammar without non-generating symbols).
   – Remove a production if a non-generating symbol appears in that production (on its right-side or its left-side)
3. Compute reachable(G_1).
4. Remove all productions containing at least one non-reachable symbol in order to create a new grammar G_2 without useless symbols (a grammar without non-reachable symbols and non-generating symbols).

The new grammar $\boldsymbol{G_2}$ (a grammar without useless symbols) will be equivalent to the original grammar $\boldsymbol{G}$.

Eliminate Useless Symbols - *Example*

$\boldsymbol{G}$: $\boldsymbol{S \rightarrow AB \mid a , A \rightarrow b}$

- **Compute generating($\boldsymbol{G}$):**
  - generating(G) $= \{a, b, A, S\}$ and **non-generating symbols are $\{\boldsymbol{B}\}$.**
- **Remove productions containing non-generating symbols:**
  - Remove $\boldsymbol{S \rightarrow AB}$ because it contains B.
  - Thus, following G_1 is a grammar without non-generating symbols.
  - G_1 is $\boldsymbol{S \rightarrow a, A \rightarrow b}$
- **Compute reachable($\boldsymbol{G_1}$):**
  - reachable(G_1)=$\{S, a\}$ and **non-reachable symbols are $\{\boldsymbol{A, b}\}$.**
- **Remove productions containing non-reachable symbols:**
  - Remove $\boldsymbol{A \rightarrow b}$ because it contains A (and/or b).
- **Grammar $\boldsymbol{G_2}$ without useless symbols** (non-generating and non-reachable symbols):

$\boldsymbol{G_2}$: $\boldsymbol{S \rightarrow a}$

Chomsky Normal Form (CNF) - *Convert the remaining rules into CNF*

- **Steps to obtain an equivalent grammar in Chomsky Normal Form:**

1. Add a new start variable S_0.
2. Eliminate ε-productions.
3. Eliminate unit productions.
4. Eliminate useless symbols.
   i. Eliminate non-generating symbols.
   ii. Eliminate non-reachable symbols.

cleanup steps

5. **Convert the remaining rules into CNF:**

   Now, to obtain a grammar in CNF, we want every rule to be the form

   $$A \rightarrow BC \qquad A \rightarrow a$$

   i. Arrange that all bodies of length 2 or more consists of only variables.

   ii. Break bodies of length 3 or more into a cascade of two-variable-bodied productions.

Chomsky Normal Form (CNF) - *Convert the remaining rules into CNF*

i. Arrange that all bodies of length 2 or more consists of only variables.
   - For every terminal a that appears in a body of length≥2, create a new variable, say X_a, and replace a by X_a in all bodies.
   - Then add a new rule $X_a \rightarrow a$.

ii. Break bodies of length 3 or more into a cascade of two-variable-bodied productions.
   - For each rule of the form

     $A \rightarrow B_1, \ldots, B_k$

   $k \geq 3$, introduce new variables $Y_1, \ldots, Y_{k-2}$ and replace the rule with

   $A \rightarrow B_1 Y_1$

   $Y_1 \rightarrow B_2 Y_2$

   …

   $Y_{k-3} \rightarrow B_{k-2} Y_{k-2}$

   $Y_{k-2} \rightarrow B_{k-1} B_k$

Chomsky Normal Form (CNF) - *Convert the remaining rules into CNF - Example*

$S_0 \rightarrow \varepsilon \mid 01 \mid 0S1$

$S \rightarrow 01 \mid 0S1$

already cleaned grammar

- Arrange that all bodies of length 2 or more consists of only variables.

  $S_0 \rightarrow \varepsilon \mid X_0 X_1 \mid X_0 S X_1$
  $S \rightarrow X_0 X_1 \mid X_0 S X_1$
  $X_0 \rightarrow 0$
  $X_1 \rightarrow 1$

- Break bodies of length 3 or more into two-variable-bodied productions.

  $S_0 \rightarrow X_0 S X_1$ ➔ $S_0 \rightarrow X_0 Y_1$ $\quad Y_1 \rightarrow S X_1$
  $S \rightarrow X_0 S X_1$ ➔ $S \rightarrow X_0 Y_1$

Grammar in CNF:

$S_0 \rightarrow \varepsilon \mid X_0 X_1 \mid X_0 Y_1$ $\quad Y_1 \rightarrow S X_1$
$S \rightarrow X_0 X_1 \mid X_0 Y_1$
$X_0 \rightarrow 0$
$X_1 \rightarrow 1$

Chomsky Normal Form (CNF) - *Converting into CNF: A Full Example*

$$S \rightarrow ABA$$
$$A \rightarrow aA \mid \varepsilon$$
$$B \rightarrow bBc \mid \varepsilon$$

Step 1. Add a new start variable S_0

$S \rightarrow ABA$
$A \rightarrow aA \mid \varepsilon$
$B \rightarrow bBc \mid \varepsilon$

$S_0 \rightarrow S$
$S \rightarrow ABA$
$A \rightarrow aA \mid \varepsilon$
$B \rightarrow bBc \mid \varepsilon$

Step 2. Eliminate ε-productions.

$nullable(G) = \{A, B, S, S_0\}$

$S_0 \rightarrow S$
$S \rightarrow ABA$
$A \rightarrow aA \mid \varepsilon$
$B \rightarrow bBc \mid \varepsilon$

$S_0 \rightarrow S \mid \varepsilon$
$S \rightarrow ABA \mid BA \mid AA \mid AB \mid A \mid B \mid A \mid \varepsilon$
$A \rightarrow aA \mid a \mid \varepsilon$
$B \rightarrow bBc \mid \varepsilon \mid bc$

$S_0 \rightarrow S \mid \varepsilon$
$S \rightarrow ABA \mid BA \mid AA \mid AB \mid A \mid B$
$A \rightarrow aA \mid a$
$B \rightarrow bBc \mid bc$

Chomsky Normal Form (CNF) - *Converting into CNF: A Full Example*

Step 3. Eliminate unit productions.

$$
\begin{aligned}
S_0 &\to S \mid \varepsilon \\
S &\to ABA \mid BA \mid AA \mid AB \mid A \mid B \\
A &\to aA \mid a \\
B &\to bBc \mid bc
\end{aligned}
$$

$$
\begin{aligned}
S_0 &\to \varepsilon \mid ABA \mid BA \mid AA \mid AB \mid aA \mid a \mid bBc \mid bc \\
S &\to ABA \mid BA \mid AA \mid AB \mid aA \mid a \mid bBc \mid bc \\
A &\to aA \mid a \\
B &\to bBc \mid bc
\end{aligned}
$$

Step 4. Eliminate useless symbols.

i. Eliminate non-generating symbols. none

ii. Eliminate non-reachable symbols. S

$$
\begin{aligned}
S_0 &\to \varepsilon \mid ABA \mid BA \mid AA \mid AB \mid aA \mid a \mid bBc \mid bc \\
S &\to ABA \mid BA \mid AA \mid AB \mid aA \mid a \mid bBc \mid bc \\
A &\to aA \mid a \\
B &\to bBc \mid bc
\end{aligned}
$$

$$
\begin{aligned}
S_0 &\to \varepsilon \mid ABA \mid BA \mid AA \mid AB \mid aA \mid a \mid bBc \mid bc \\
A &\to aA \mid a \\
B &\to bBc \mid bc
\end{aligned}
$$

Chomsky Normal Form (CNF) - *Converting into CNF: A Full Example*

Step 5. Convert the remaining rules into CNF:

Arrange that all bodies of length 2 or more consists of only variables.

$$S_0 \rightarrow \varepsilon \mid ABA \mid BA \mid AA \mid AB \mid aA \mid a \mid bBc \mid bc$$
$$A \rightarrow aA \mid a$$
$$B \rightarrow bBc \mid bc$$

$$S_0 \rightarrow \varepsilon \mid ABA \mid BA \mid AA \mid AB \mid XA \mid a \mid YBZ \mid YZ$$
$$A \rightarrow XA \mid a$$
$$B \rightarrow YBZ \mid YZ$$
$$X \rightarrow a$$
$$Y \rightarrow b$$
$$Z \rightarrow c$$

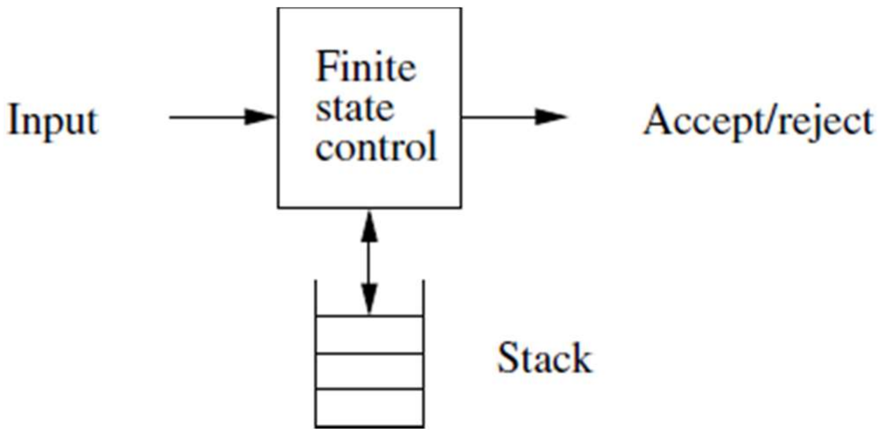
Break bodies of length 3 or more into two-variable-bodied productions.

$$S_0 \rightarrow \varepsilon \mid ABA \mid BA \mid AA \mid AB \mid XA \mid a \mid YBZ \mid YZ$$
$$A \rightarrow XA \mid a$$
$$B \rightarrow YBZ \mid YZ$$
$$X \rightarrow a$$
$$Y \rightarrow b$$
$$Z \rightarrow c$$

$$S_0 \rightarrow \varepsilon \mid AC \mid BA \mid AA \mid AB \mid XA \mid a \mid YD \mid YZ$$
$$C \rightarrow BA$$
$$A \rightarrow XA \mid a$$
$$B \rightarrow YD \mid YZ$$
$$D \rightarrow BZ$$
$$X \rightarrow a \quad Y \rightarrow b \quad Z \rightarrow c$$

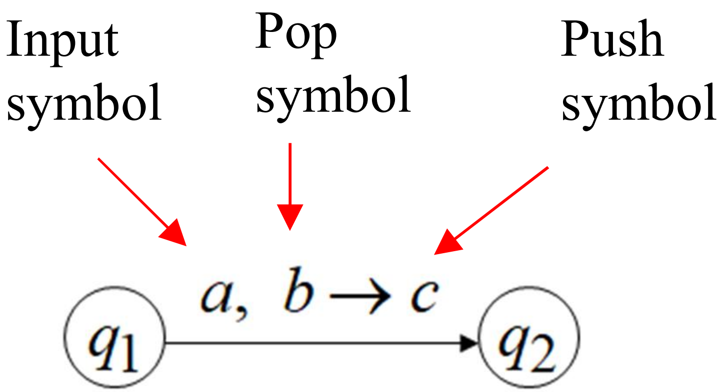
Pushdown Automata

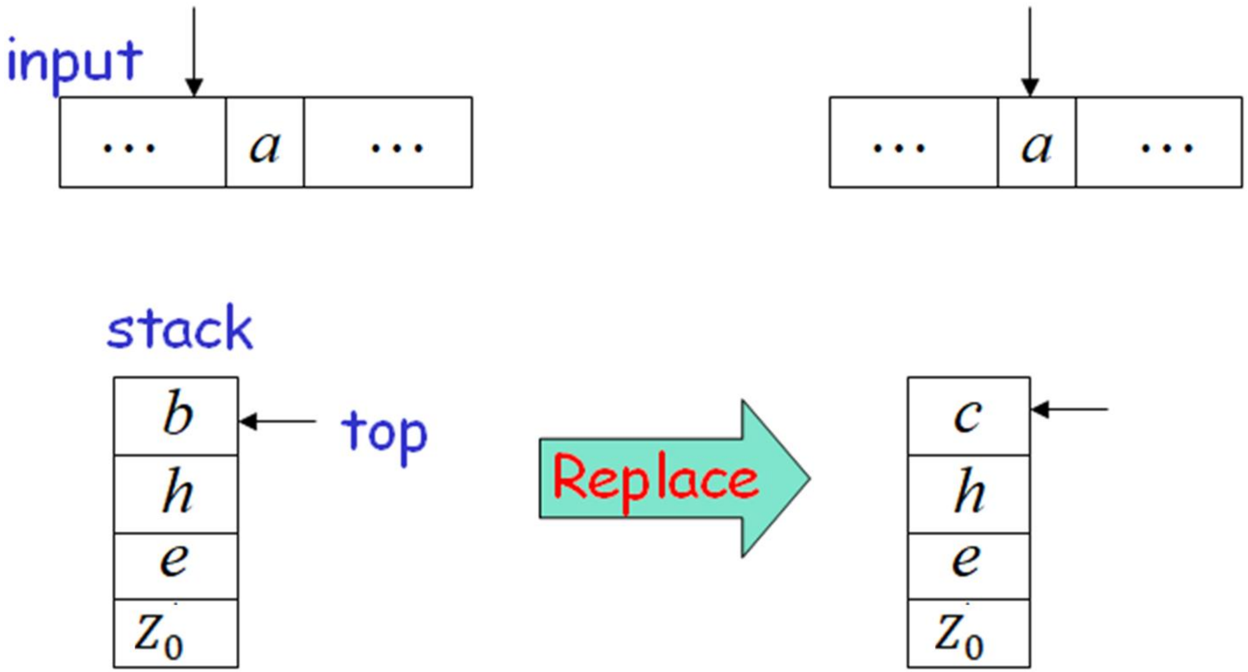
- **Pushdown automata** accept exactly **context free languages**.
- A **pushdown automaton (PDA)** is essentially **an NFA with a stack**.
- **On a transition, a PDA:**
  1. **Consumes an input symbol (or ε-transition).**
  2. **Goes to a new state (or stays in the old).**
  3. **Replaces the top of the stack item by any string (does nothing, pops the stack, or pushes a string onto the stack)**



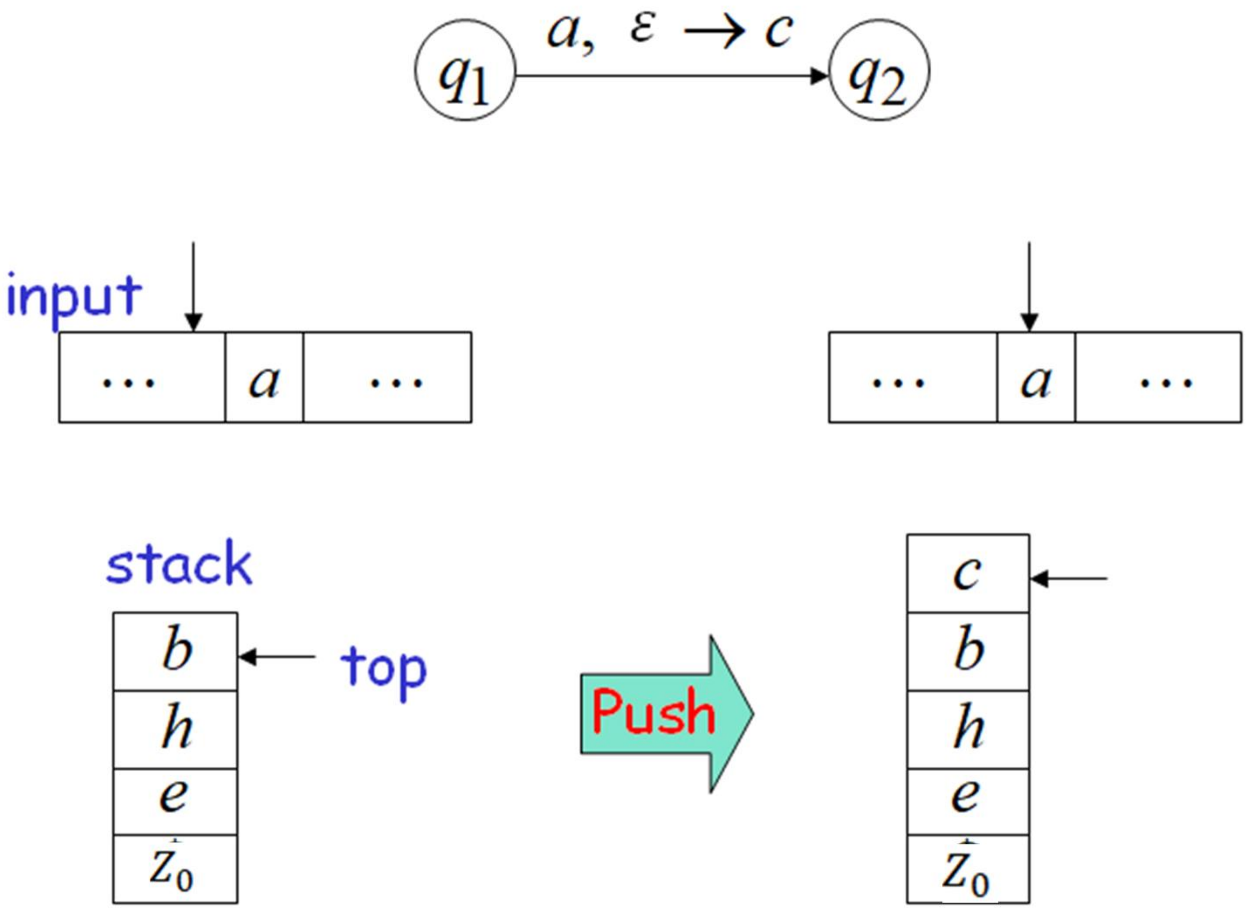
Pushdown Automata - Transitions

- No explicit mechanism to test for an empty stack.
- Initially place a special symbol Z_0 on the stack.
- If it ever sees the Z_0, it knows that the stack is empty.

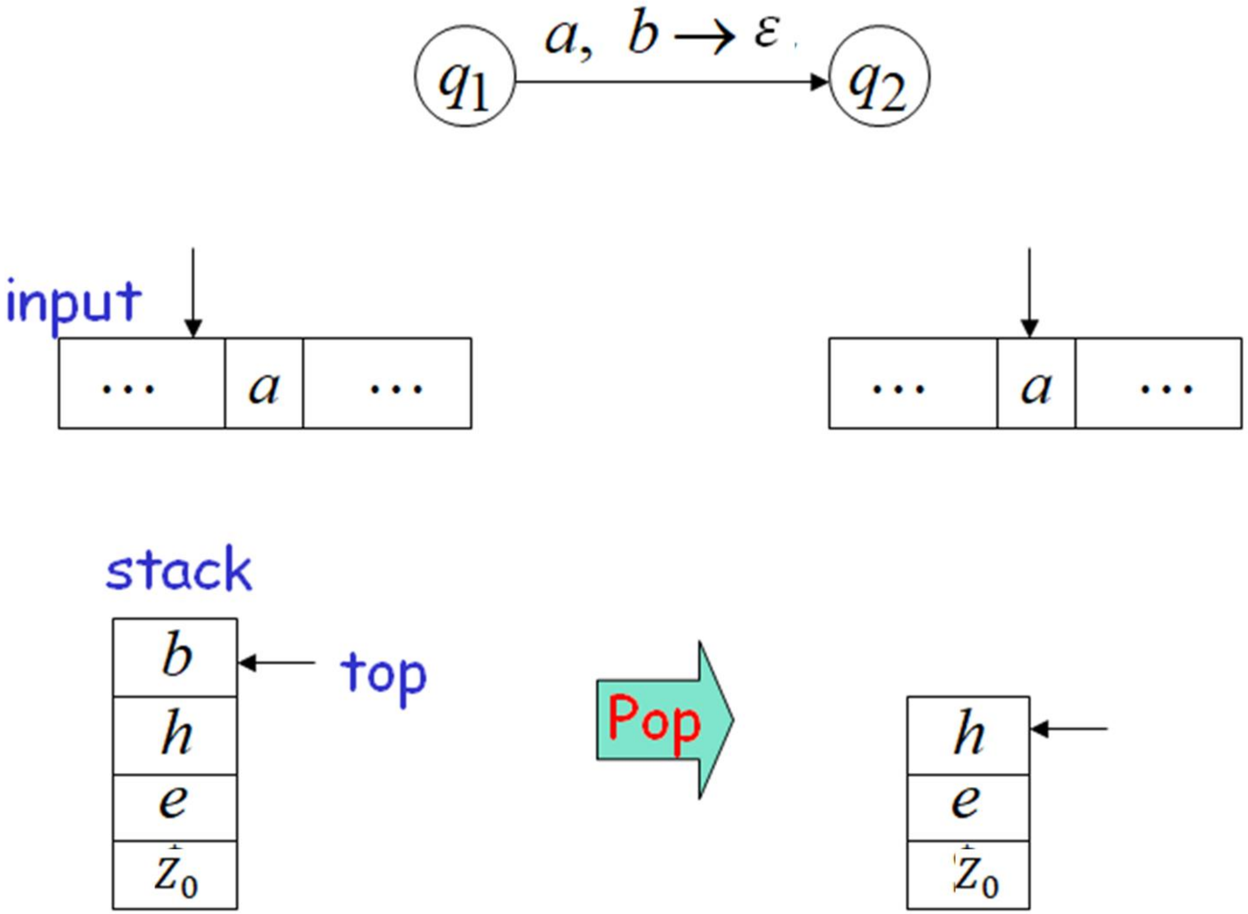




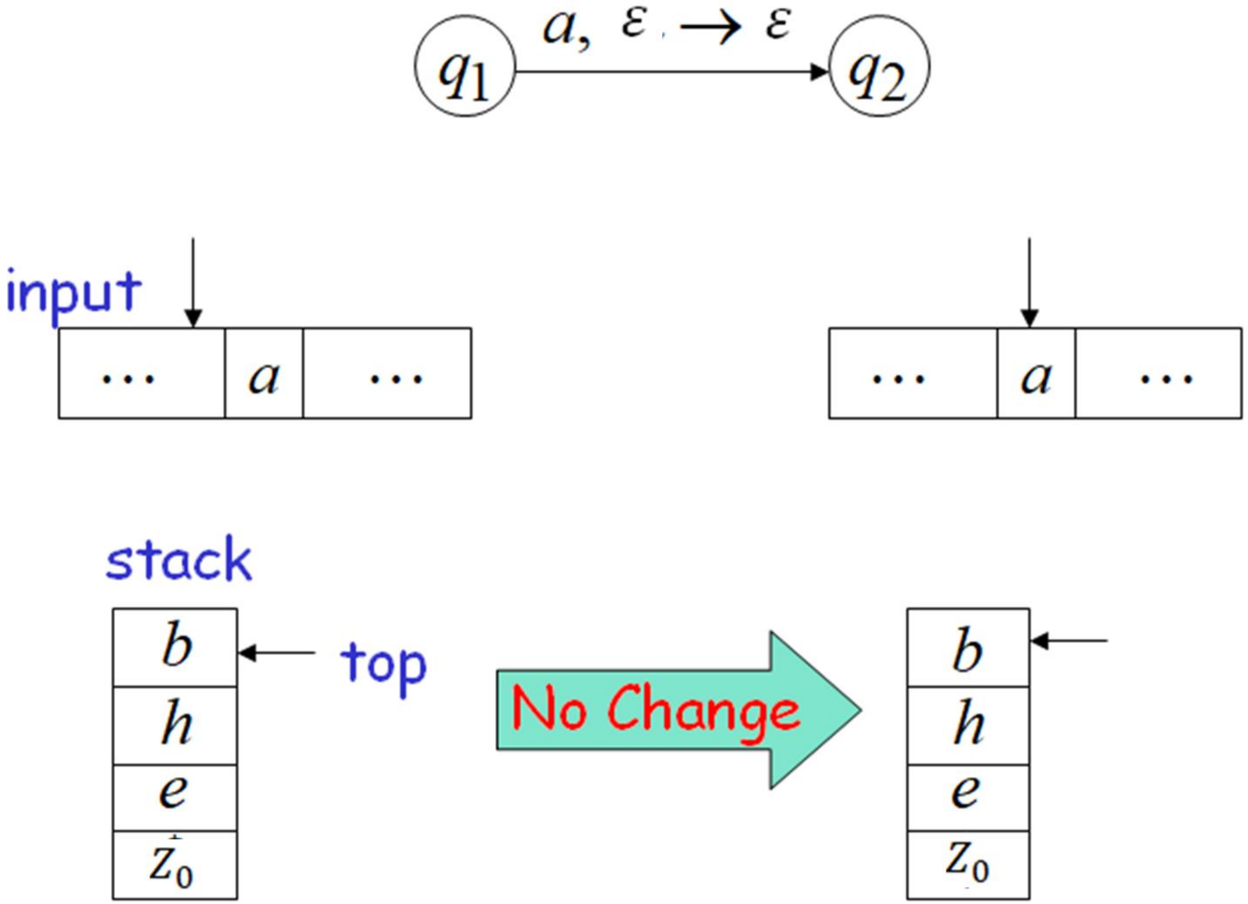
Pushdown Automata - Transitions



Pushdown Automata - Transitions



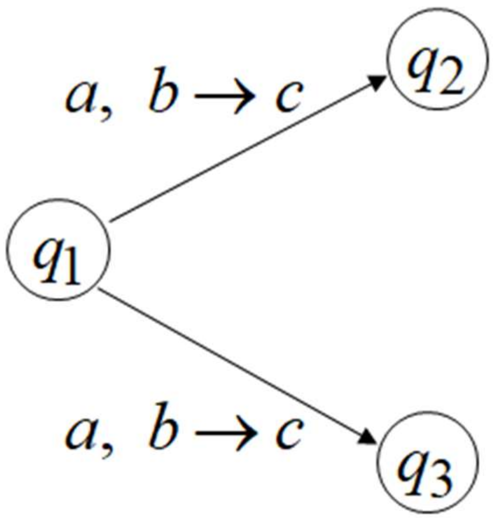
Pushdown Automata - Transitions

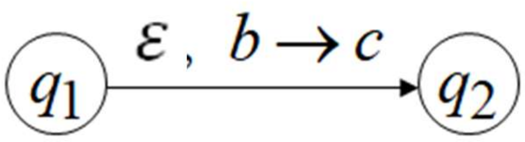


Pushdown Automata - Transitions

- PDAs are non-deterministic

- Allowed non-deterministic transitions



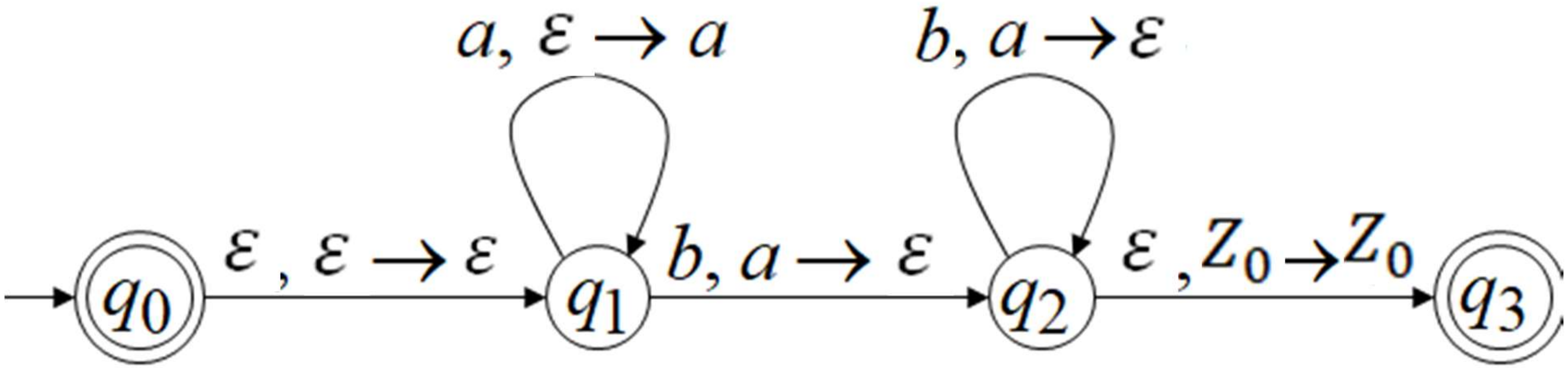


ε – transition

Pushdown Automata - Transitions

- Example PDA M

$$L(M) = \{a^n b^n : n \geq 0\}$$



Pushdown Automata - Transitions

$$L(M) = \{a^n b^n : n \geq 0\}$$

1.Push the a's

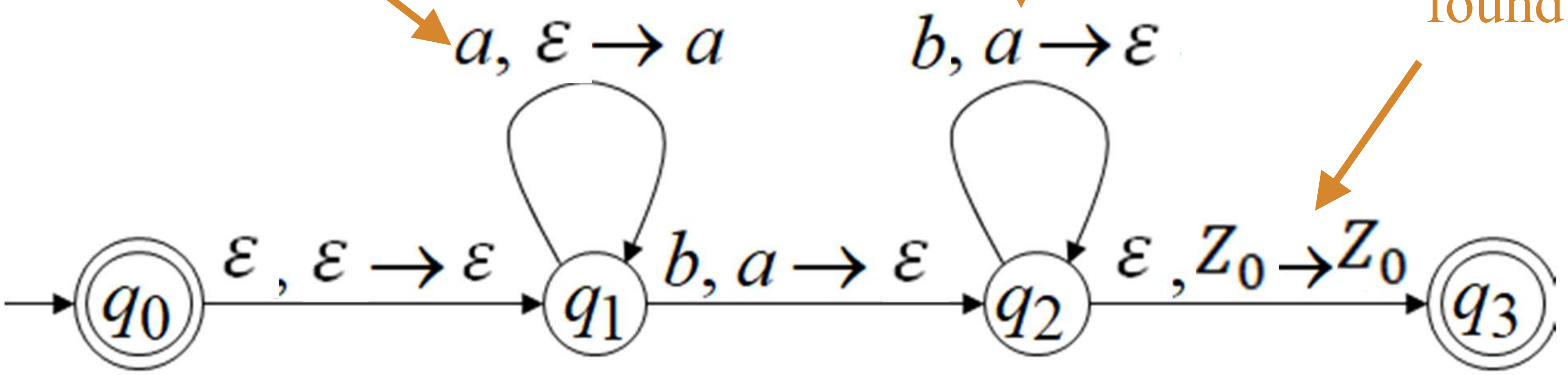
on the stack

2. Match the b's on input

with a's on stack

3. Match

found



PDA - Acceptance

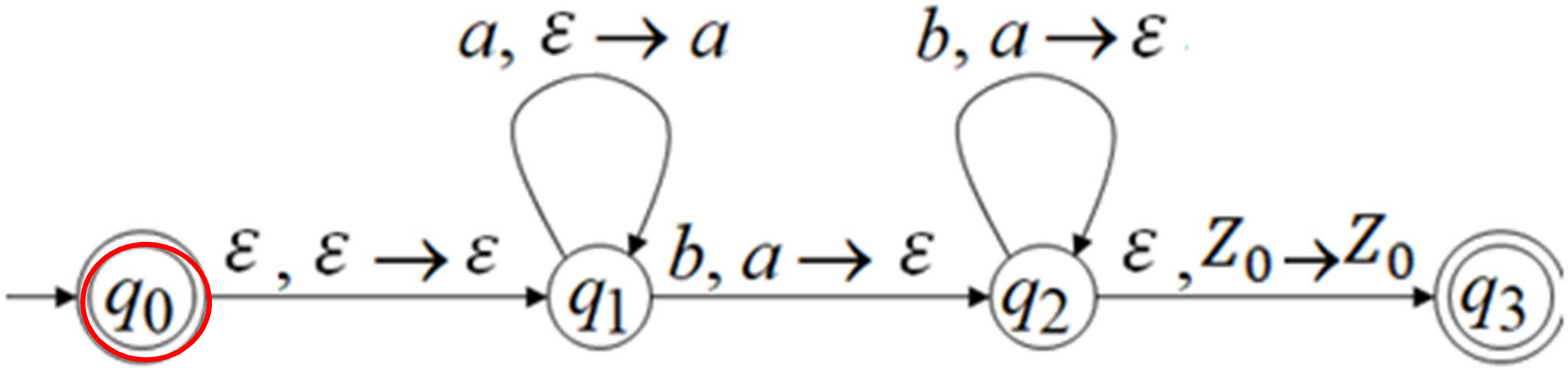
Time 0

Input

a	a	a	b	b	b

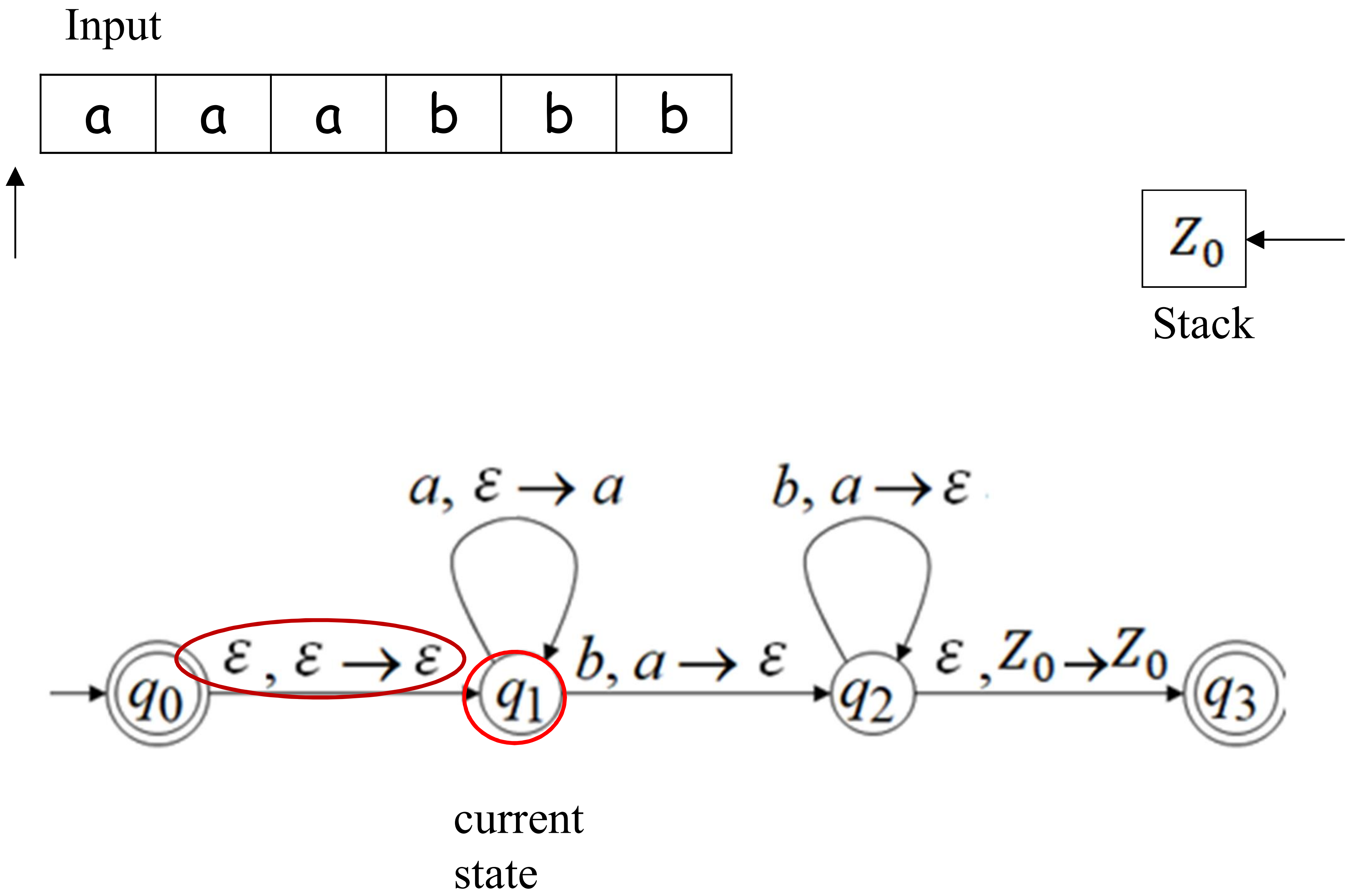
Z_0

Stack



current
state

Time 1

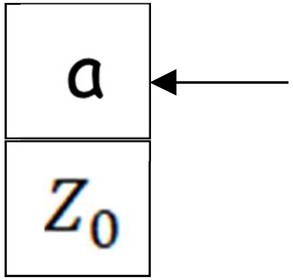


Input
a a a b b b
Z_0
Stack
$a, \varepsilon \rightarrow a$
$b, a \rightarrow \varepsilon$
$\varepsilon, \varepsilon \rightarrow \varepsilon$
$b, a \rightarrow \varepsilon$
$\varepsilon, Z_0 \rightarrow Z_0$
q_0
q_1
q_2
q_3
current state

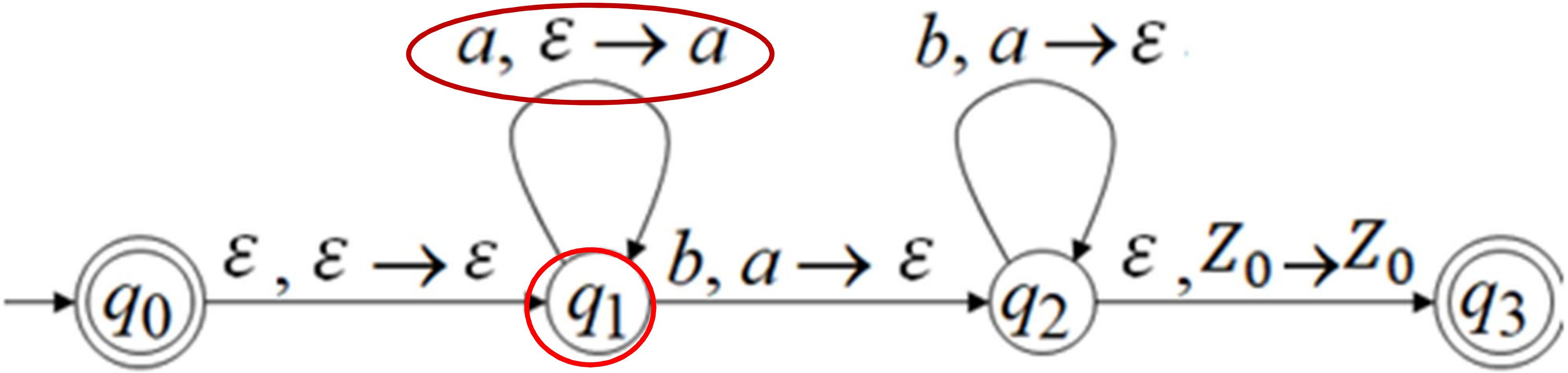
Time 2

Input

a	a	a	b	b	b

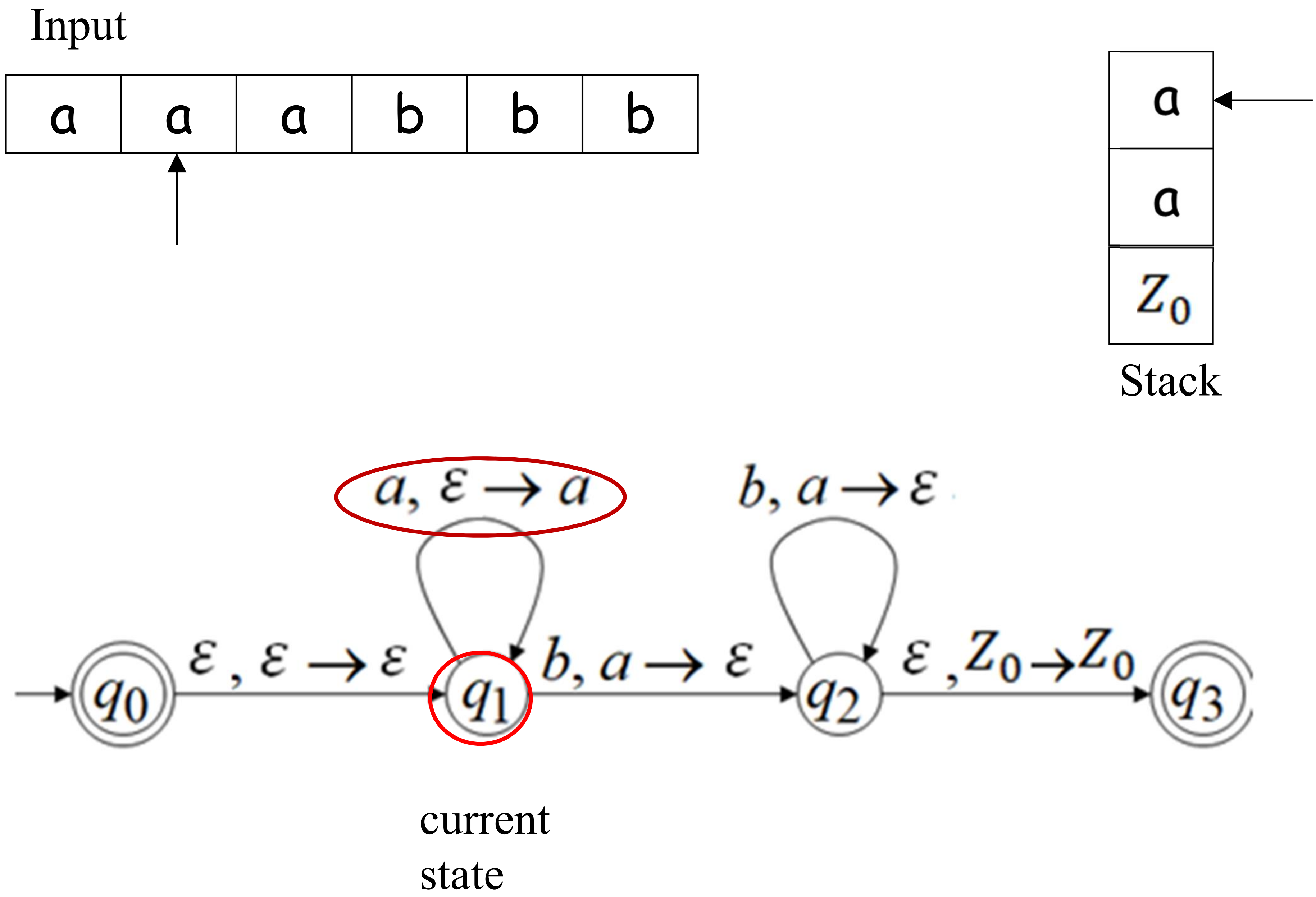


Stack

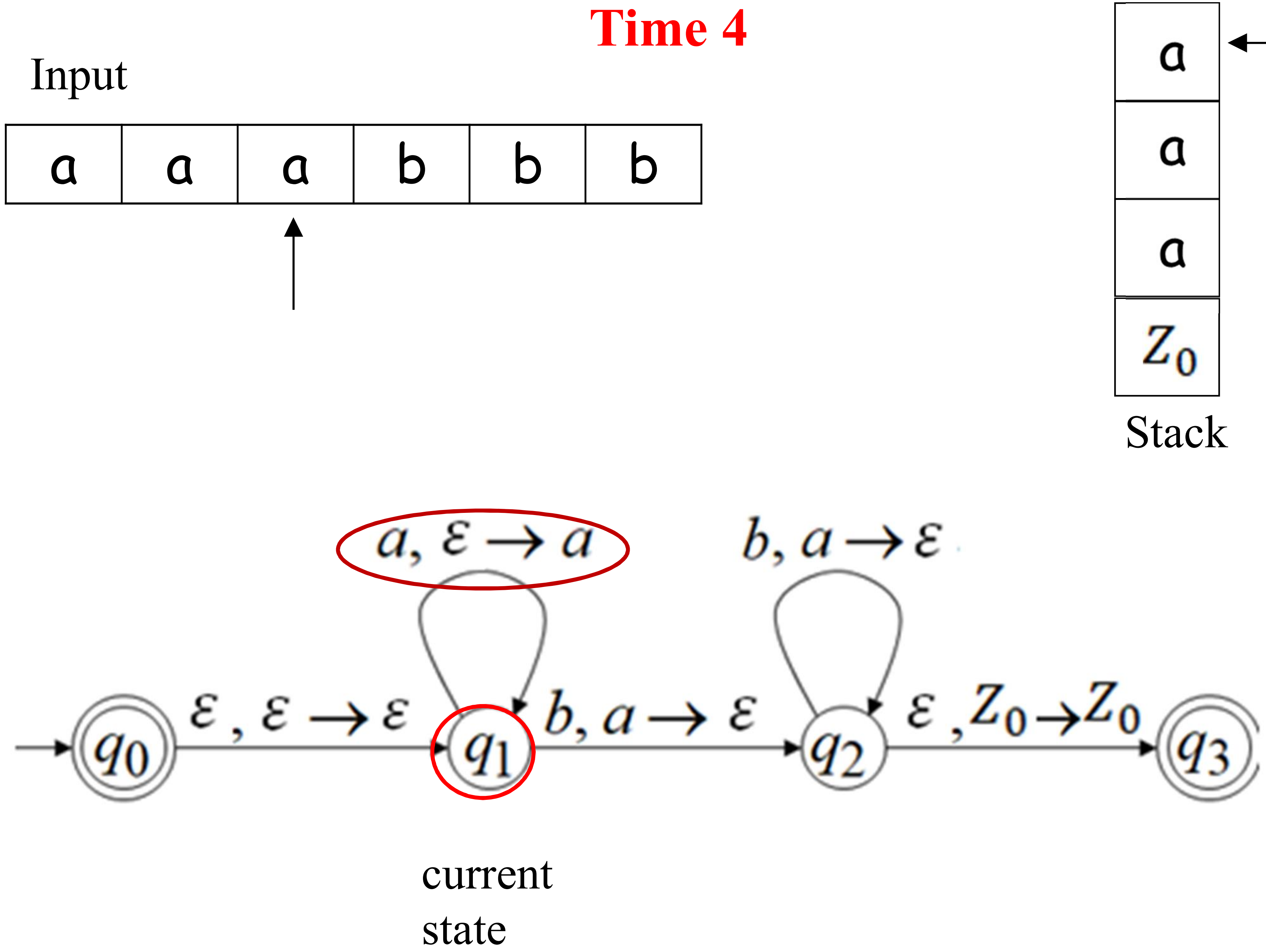


current
state

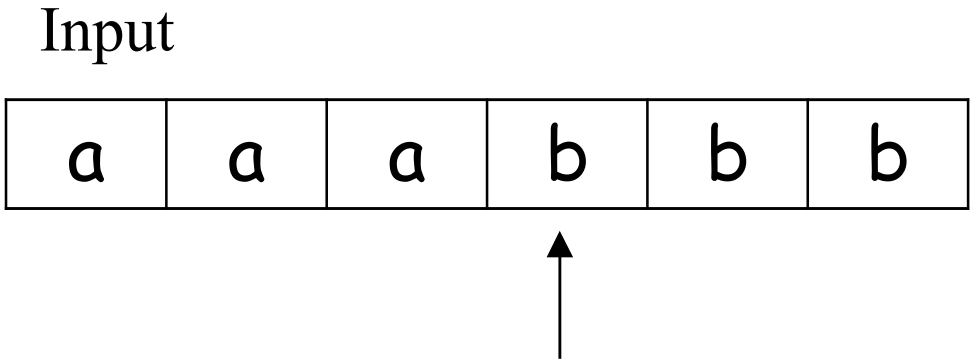
Time 3


Input
a a a b b b
a
a
Z_0
Stack
$a, \varepsilon \to a$
$b, a \to \varepsilon$
$\varepsilon, \varepsilon \to \varepsilon$
$b, a \to \varepsilon$
$\varepsilon, Z_0 \to Z_0$
q_0
q_1
q_2
q_3
current
state

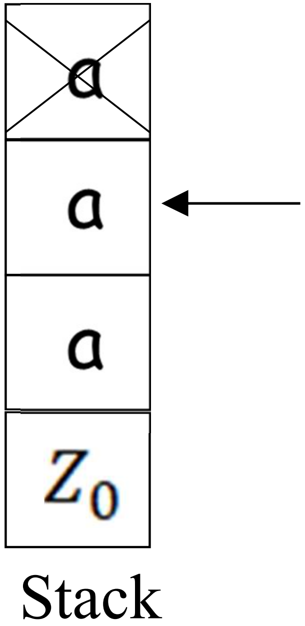
Time 4


Input
a a a b b b
a
a
a
Z_0
Stack
$a, \varepsilon \to a$
$b, a \to \varepsilon$
$\varepsilon, \varepsilon \to \varepsilon$
$b, a \to \varepsilon$
$\varepsilon, Z_0 \to Z_0$
q_0
q_1
q_2
q_3
current
state

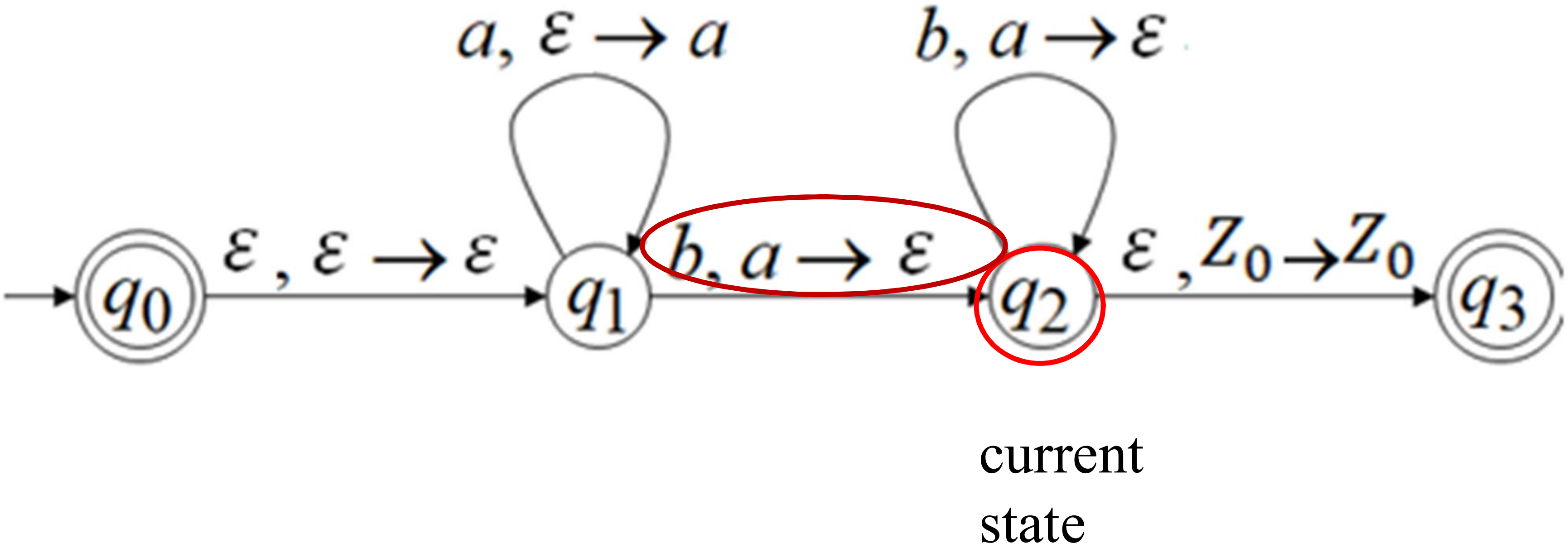
Time 5



Input
a a a b b b

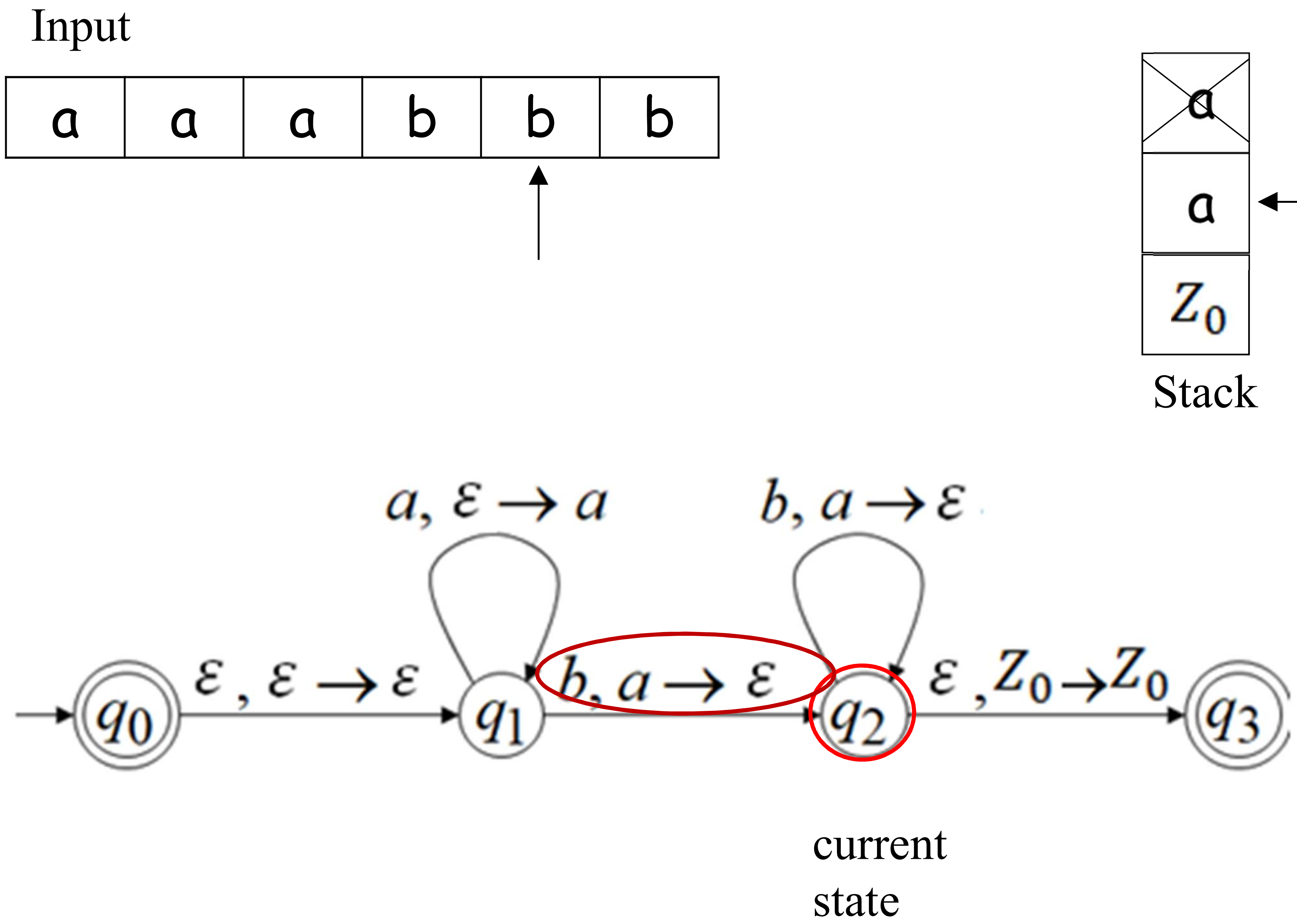


a
a
a
Z_0
Stack



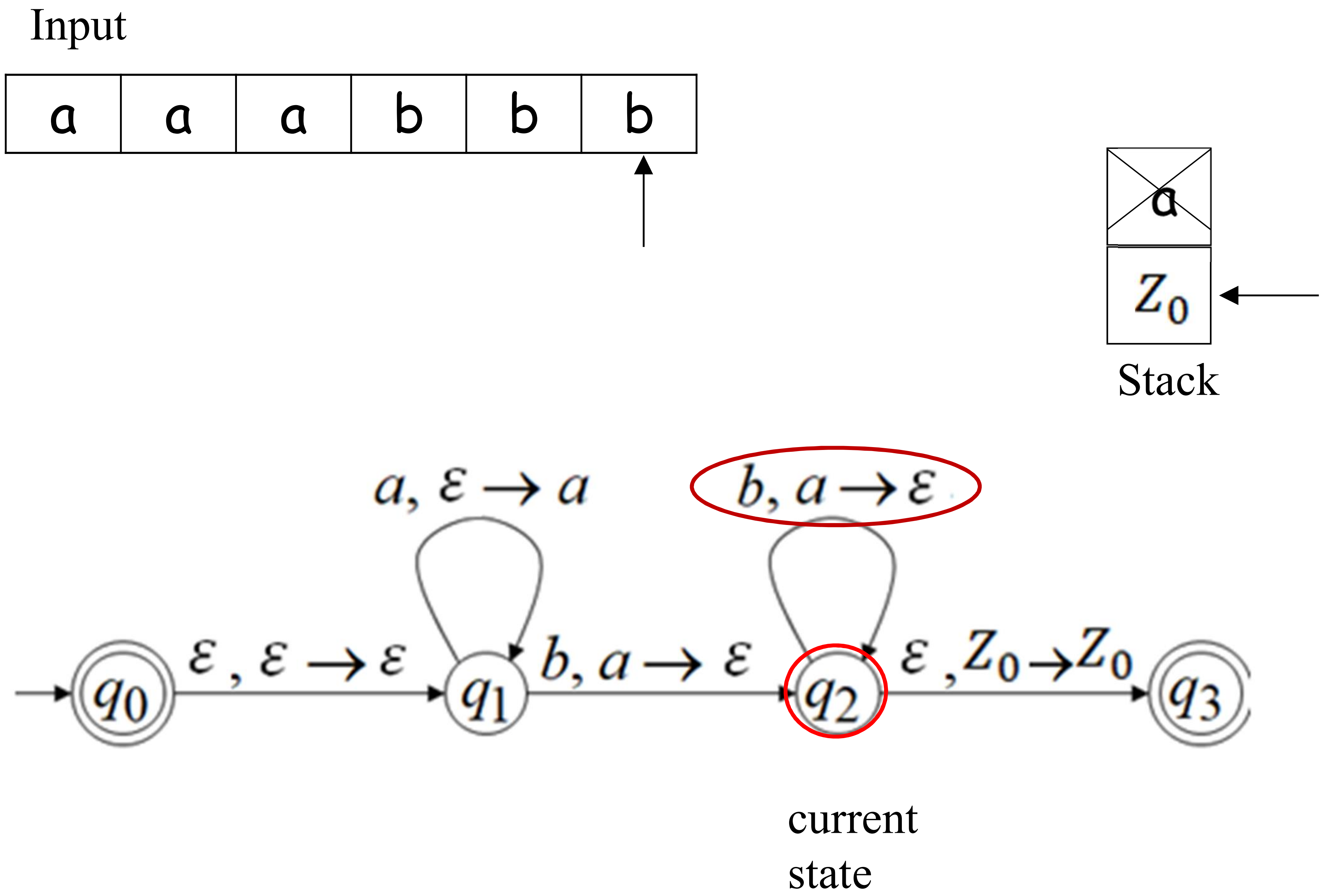
$a, \varepsilon \rightarrow a$
$b, a \rightarrow \varepsilon$
$\varepsilon, \varepsilon \rightarrow \varepsilon$
$b, a \rightarrow \varepsilon$
$\varepsilon, Z_0 \rightarrow Z_0$
q_0
q_1
q_2
q_3
current state

Time 6



Input
a a a b b b
Stack
a
a
Z_0
$a, \varepsilon \to a$
$b, a \to \varepsilon$
$\varepsilon, \varepsilon \to \varepsilon$
$b, a \to \varepsilon$
$\varepsilon, Z_0 \to Z_0$
q_0
q_1
q_2
q_3
current state

Time 7



Input
a
a
a
b
b
b
a
Z_0
Stack
$a, \varepsilon \to a$
$b, a \to \varepsilon$
$\varepsilon, \varepsilon \to \varepsilon$
$b, a \to \varepsilon$
$\varepsilon, Z_0 \to Z_0$
q_0
q_1
q_2
q_3
current
state

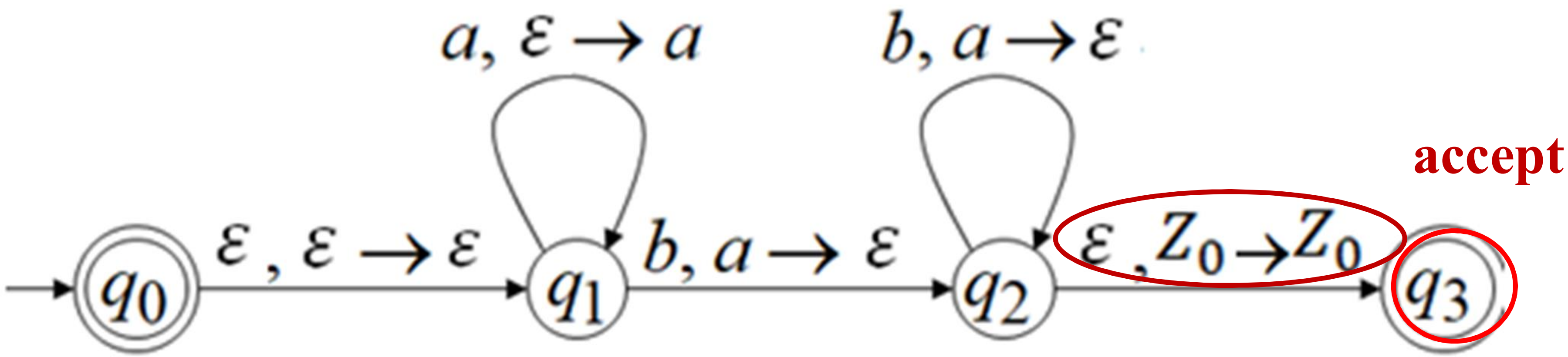
Time 8

Input

a	a	a	b	b	b

Stack

Z_0



current state

Pushdown Automata – Acceptance

- A string is accepted if there is a computation such that:

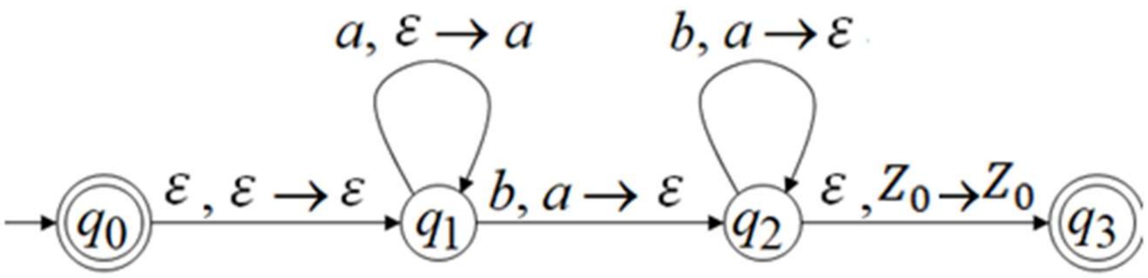
All the input is consumed

AND

The last state is an accepting state

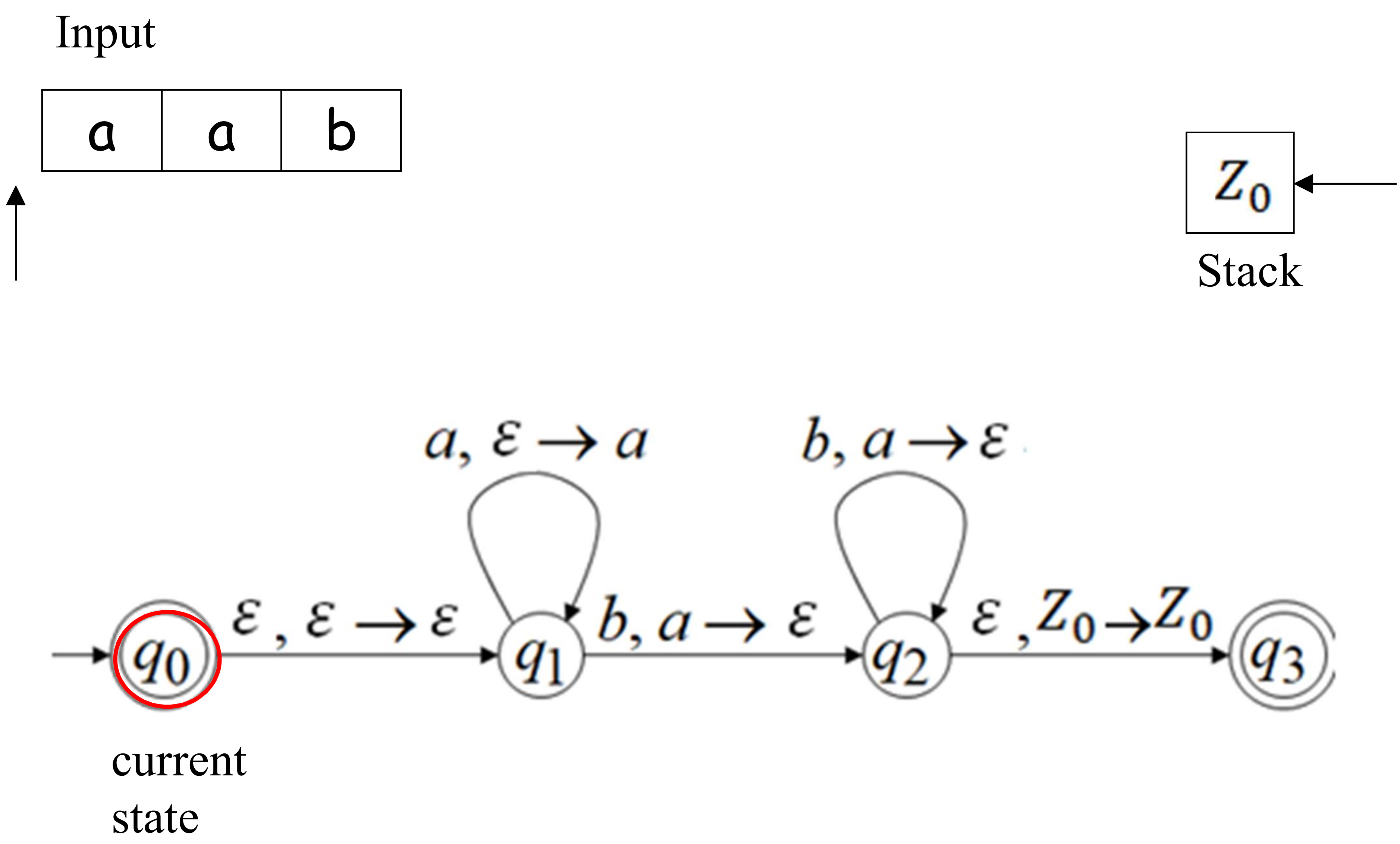
- At the end of the computation, we do not care about the stack contents (the stack can be empty at the last state)

The input string **aaabbb** is accepted by the PDA:



PDA - Rejection

Time 0



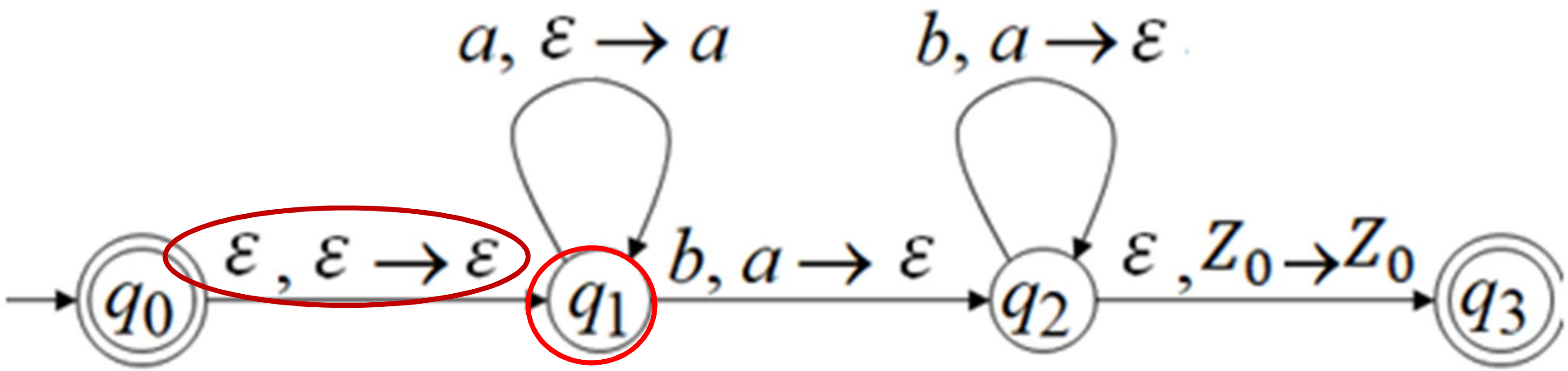
Time 1

Input

a	a	b

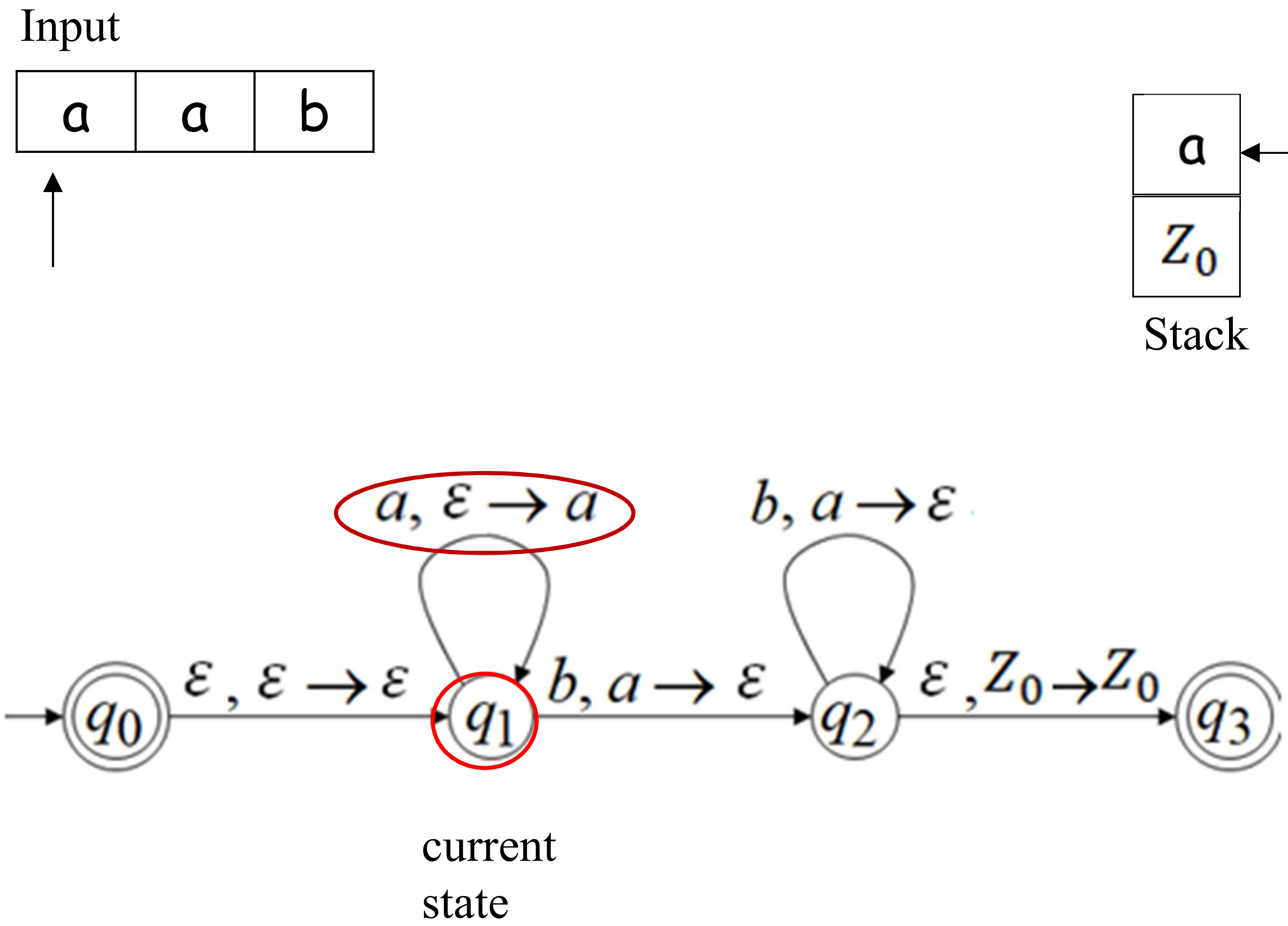
Z_0

Stack



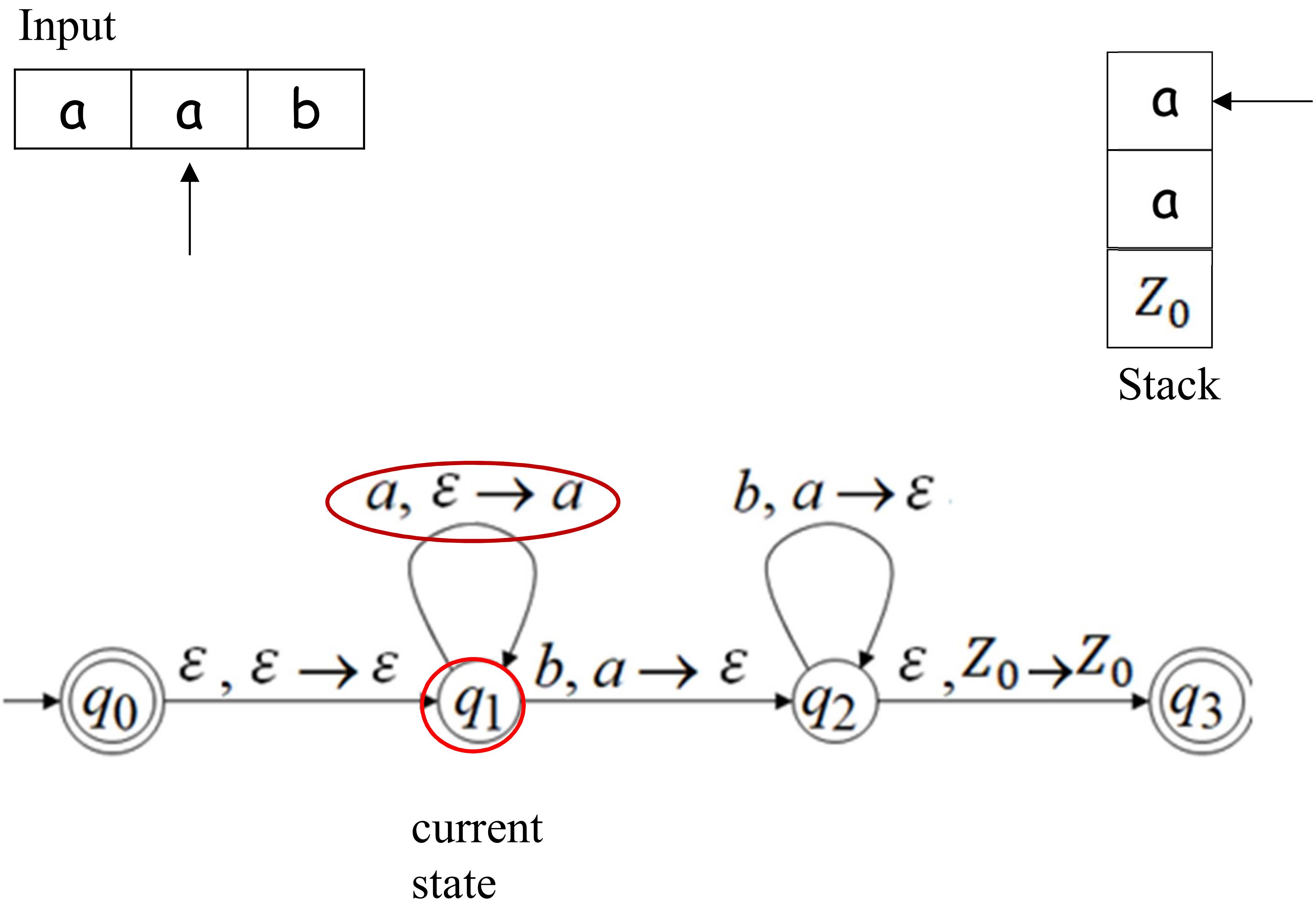
current
state

Time 2

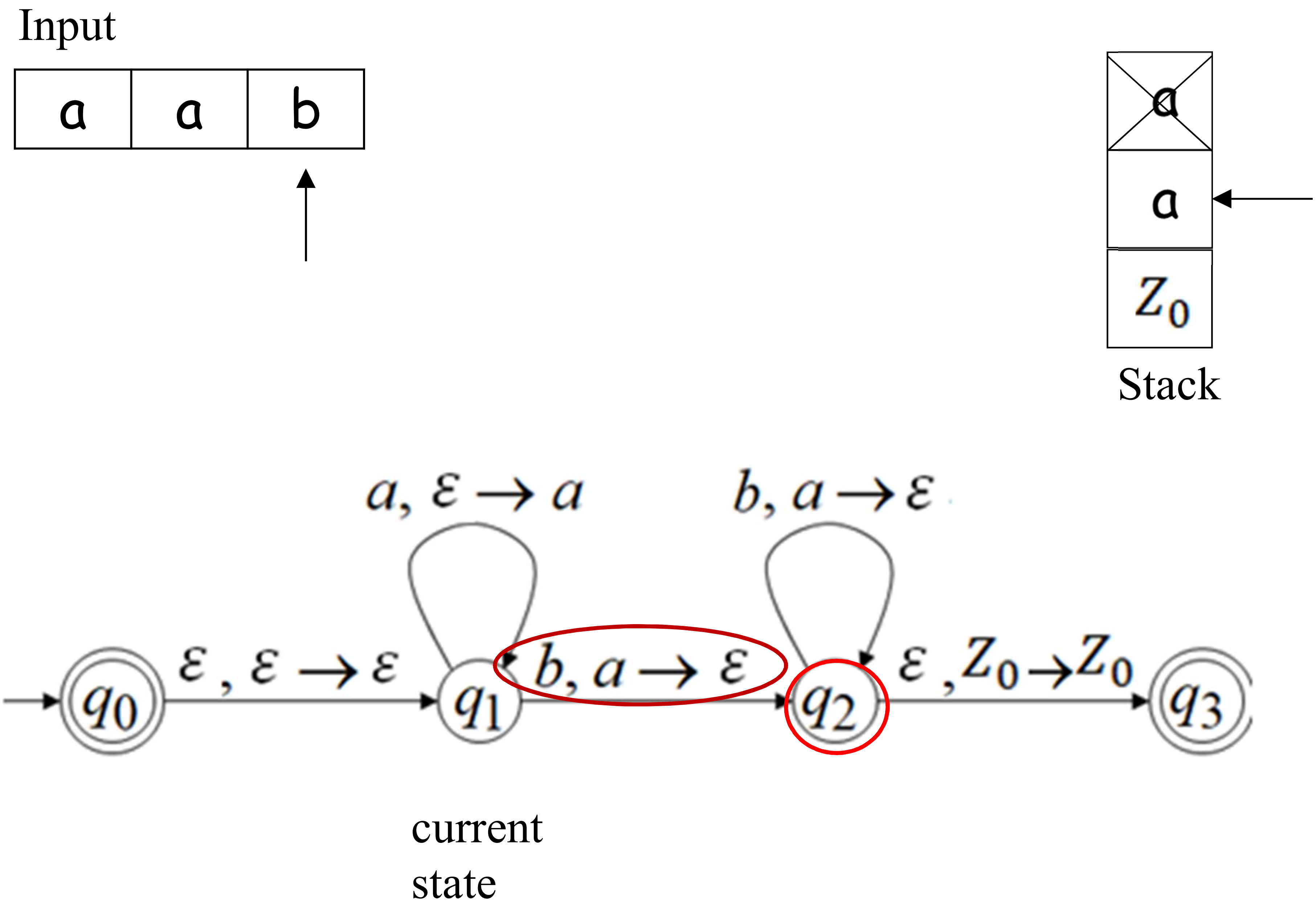


Input
a
a
b
a
Z_0
Stack
$a, \varepsilon \rightarrow a$
$b, a \rightarrow \varepsilon$
$\varepsilon, \varepsilon \rightarrow \varepsilon$
$b, a \rightarrow \varepsilon$
$\varepsilon, Z_0 \rightarrow Z_0$
q_0
q_1
q_2
q_3
current state

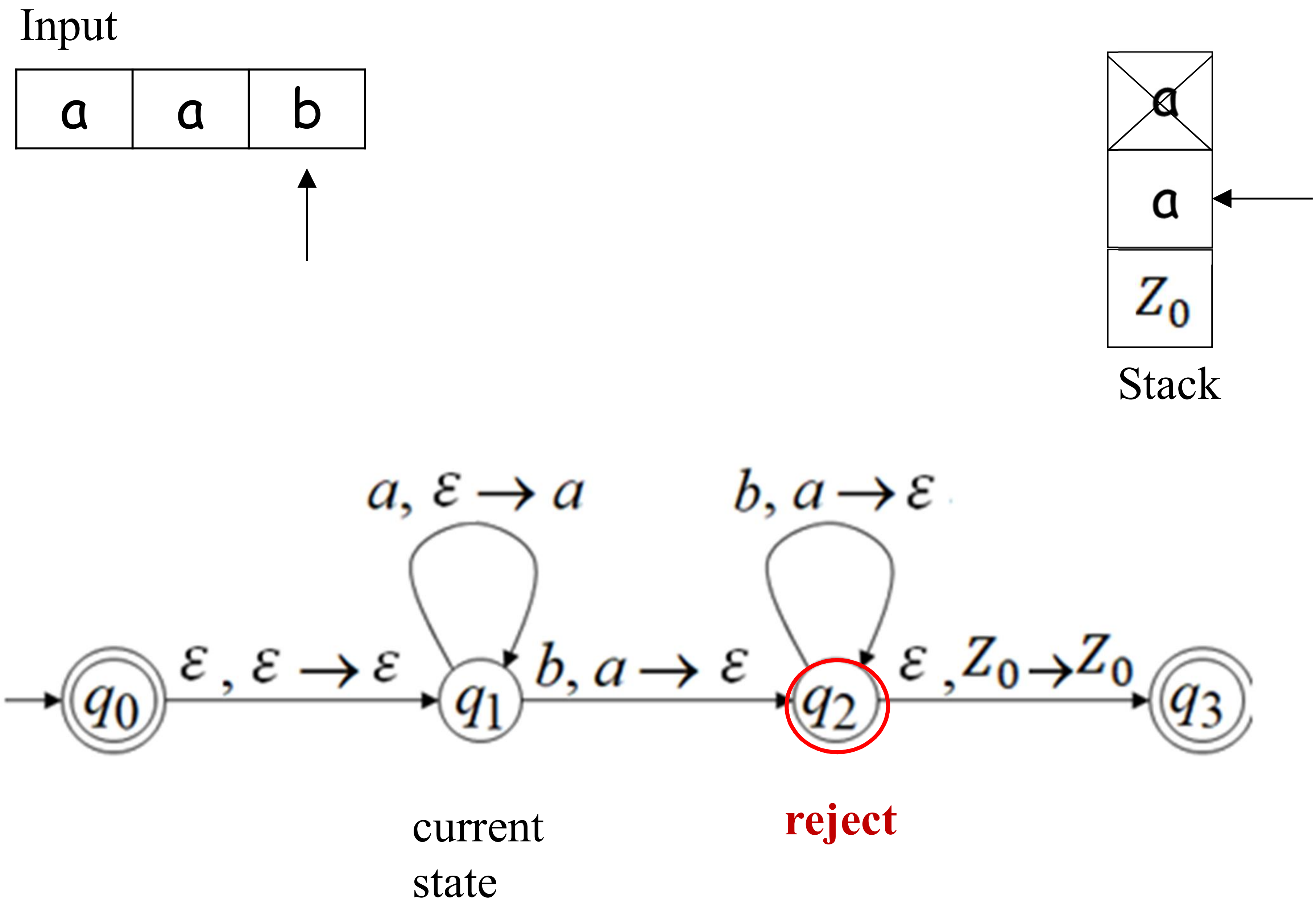
Time 3


Input
a
a
b
a
a
Z_0
Stack
$a, \varepsilon \rightarrow a$
$b, a \rightarrow \varepsilon$
$\varepsilon, \varepsilon \rightarrow \varepsilon$
$b, a \rightarrow \varepsilon$
$\varepsilon, Z_0 \rightarrow Z_0$
q_0
q_1
q_2
q_3
current
state

Time 4


Input
a
a
b
a
a
Z_0
Stack
$a, \varepsilon \rightarrow a$
$b, a \rightarrow \varepsilon$
$\varepsilon, \varepsilon \rightarrow \varepsilon$
$b, a \rightarrow \varepsilon$
$\varepsilon, Z_0 \rightarrow Z_0$
q_0
q_1
q_2
q_3
current
state

Time 4


Input
a
a
b
a
a
Z_0
Stack
$a, \varepsilon \to a$
$b, a \to \varepsilon$
$\varepsilon, \varepsilon \to \varepsilon$
$b, a \to \varepsilon$
$\varepsilon, Z_0 \to Z_0$
q_0
q_1
q_2
q_3
current state
reject

Pushdown Automata – Formal Definition

A pushdown Automata (PDA) is a seven-tuple:

$$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F),$$

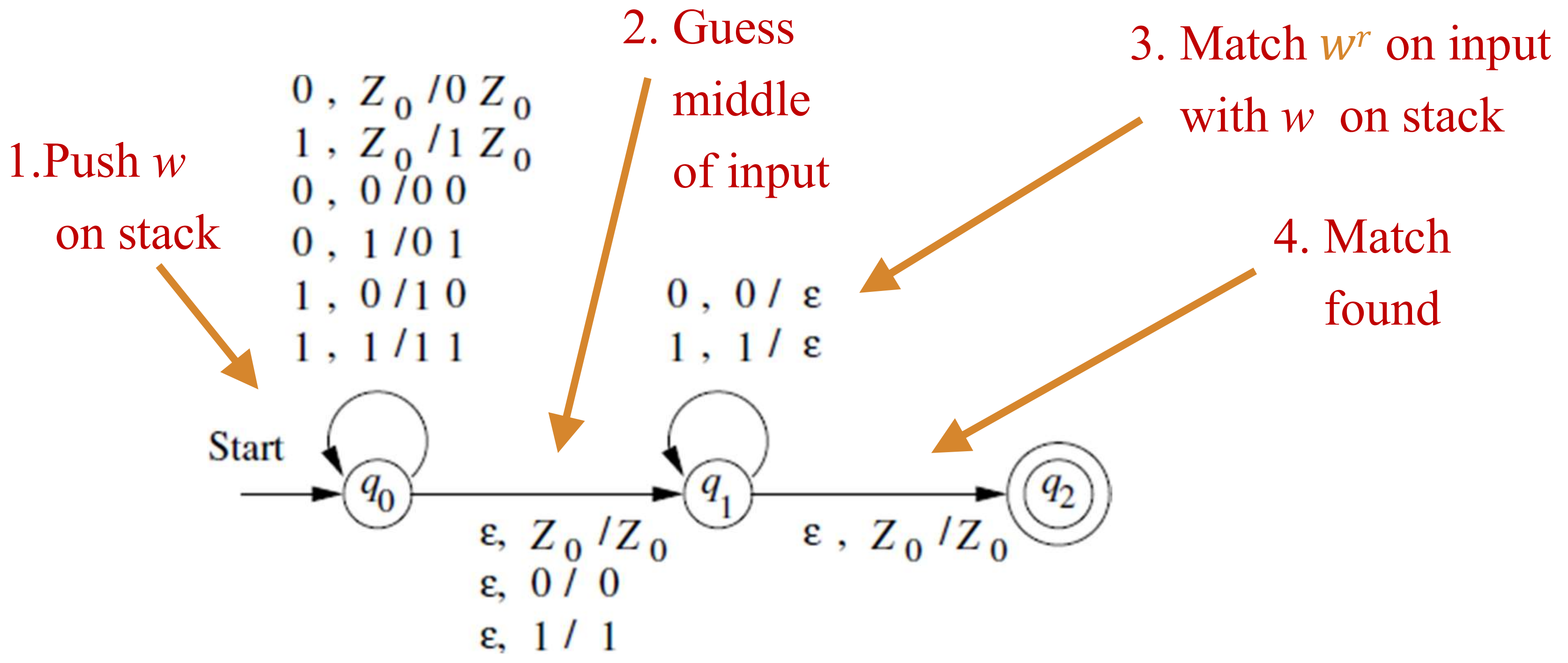
where

- Q is a finite set of states,
- Σ is a finite input alphabet,
- Γ is finite stack alphabet,
- $\delta : Q \; x \; \Sigma \cup \{\varepsilon\} \rightarrow 2^{Q x \Gamma^*}$ is the transition function,
- q_0 is a start state,
- Z_0 is the start symbol for the stack, and
- F is the set of accepting states.

Pushdown Automata – Example

- Consider a CFL $L_{ww^r} = \{ ww^r : w \in \{0,1\}^* \}$
- A pushdown automaton P for the language L_{ww^r} is as follows:

$$P = (\{q_0, q_1, q_2\}, \{0,1\}, \{0,1,Z_0\}, \delta, q_0, Z_0, \{q_2\})$$

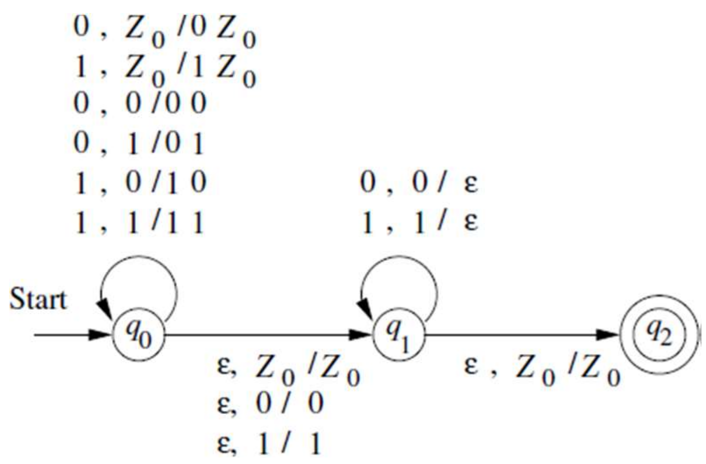


Pushdown Automata – *Table Representation of Transition Function*

- Consider a CFL $L_{ww^r} = \{ ww^r : w \in \{0,1\}* \}$
- A pushdown automaton P for the language $L_{ww^r} = \{ ww^r : w \in \{0,1\}* \}$ is as actually a seven tuple:

$$P = (\{q_0, q_1, q_2\}, \{0,1\}, \{0,1, Z_0\}, \delta, q_0, Z_0, \{q_2\})$$

where its transition function can be also shown by a table.



	$0, Z_0$	$1, Z_0$	0,0	0,1	1,0	1,1	ϵ, Z_0	$\epsilon, 0$	$\epsilon, 1$
$\rightarrow q_0$	$q_0, 0Z_0$	$q_0, 1Z_0$	$q_0, 00$	$q_0, 01$	$q_0, 10$	$q_0, 11$	q_1, Z_0	$q_1, 0$	$q_1, 1$
q_1			q_1, ϵ			q_1, ϵ	q_2, Z_0		
$\star q_2$									

Instantaneous Descriptions (ID) of a PDA

- A PDA goes from configuration to configuration when consuming input.

- **The configuration of a PDA is represented by a triple (q, w, γ) where**
  - **q is a the state,**
  - **w is the remaining input, and**
  - **γ is the stack contents**

- **A configuration triple is called an instantaneous description, or ID, of the pushdown automata.**

$$(\underset{\text{Current state}}{q},\ \underset{\text{Remaining input}}{w},\ \underset{\text{Current stack contents}}{\gamma})$$

PDA Move - turnstile $\vdash$ and $\vdash^*$ Notation

- We need a notation that describes changes in the state, the input, and stack.
- Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a PDA. We define a **move** $\vdash$ as follows.

- **If $(p, \alpha) \in \delta(q, a, X)$ where $q \in Q, a \in \Sigma \cup \{\varepsilon\}, X \in \Gamma$ and $\alpha \in \Gamma^*$. Then for all strings $w \in \Sigma^*$ and $\beta \in \Gamma^*$, we have a move (transition):**

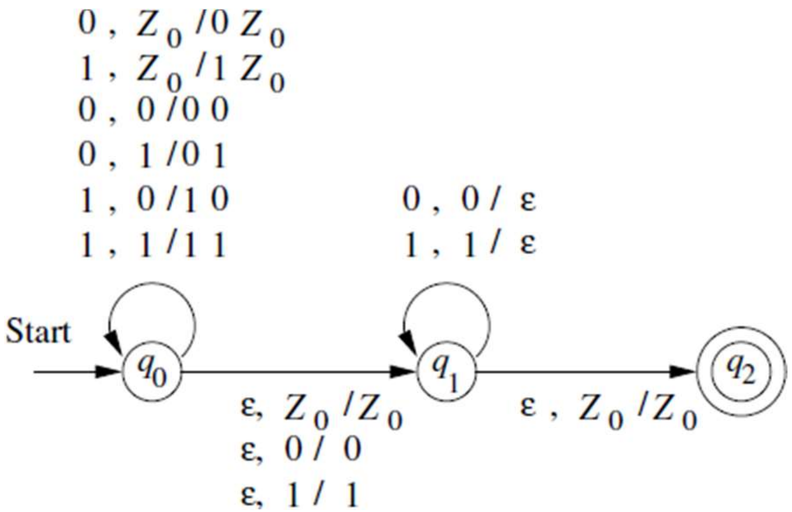
$$(q, aw, X\beta) \vdash (p, w, \alpha\beta))$$

- This move reflects the idea that, by consuming a (which may be ε) from the input and replacing X on top of the stack by α we can go from state q to state p.
  - Note that what remains on the input, w, and what is below the top of the stack, β ,do not influence the action of the PDA, they are merely carried along.

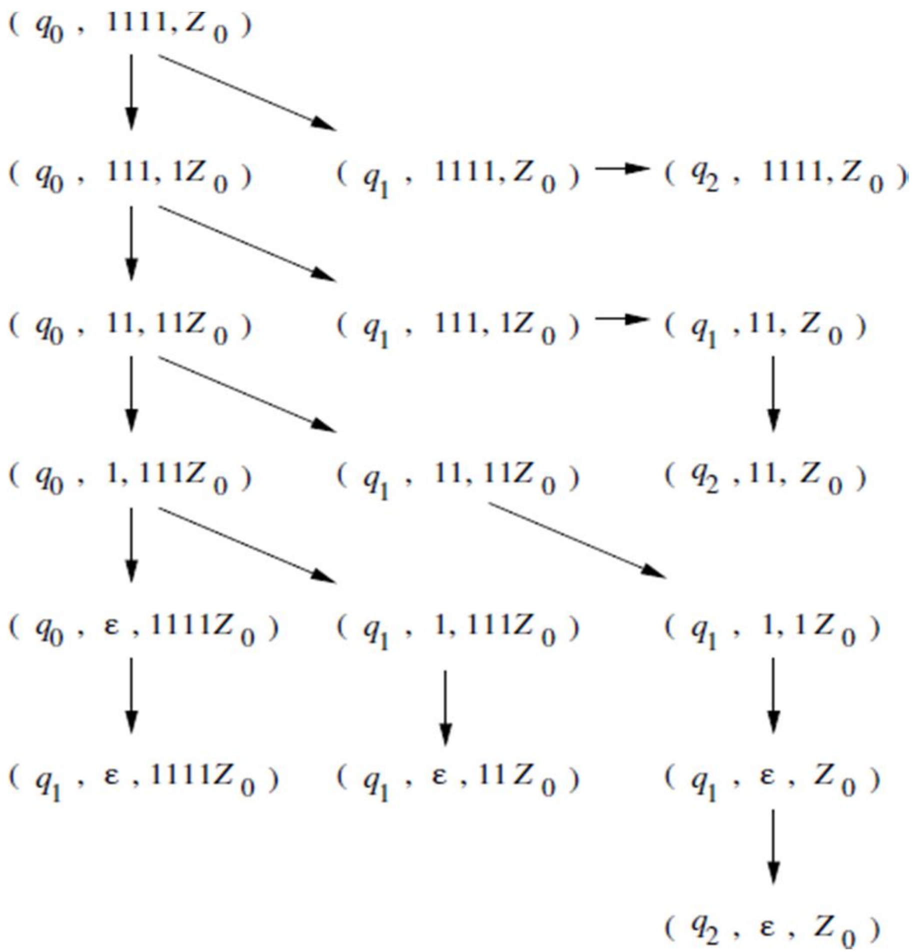
- To represent a sequence of zero or more moves of the PDA, we use $\vdash^*$.
- **A sequence of moves is also called as computation.**

PDA Move - Example

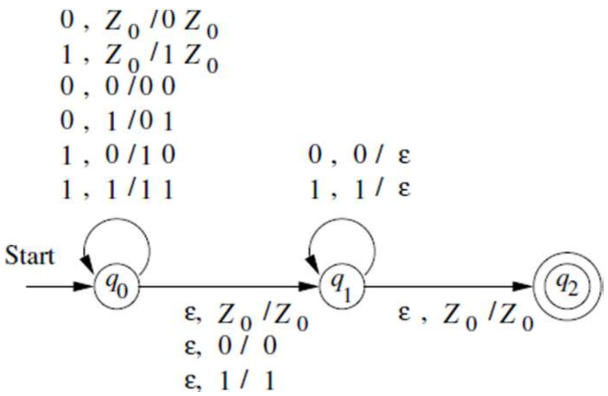
- On input 1111 the PDA has the following **computation sequences**:

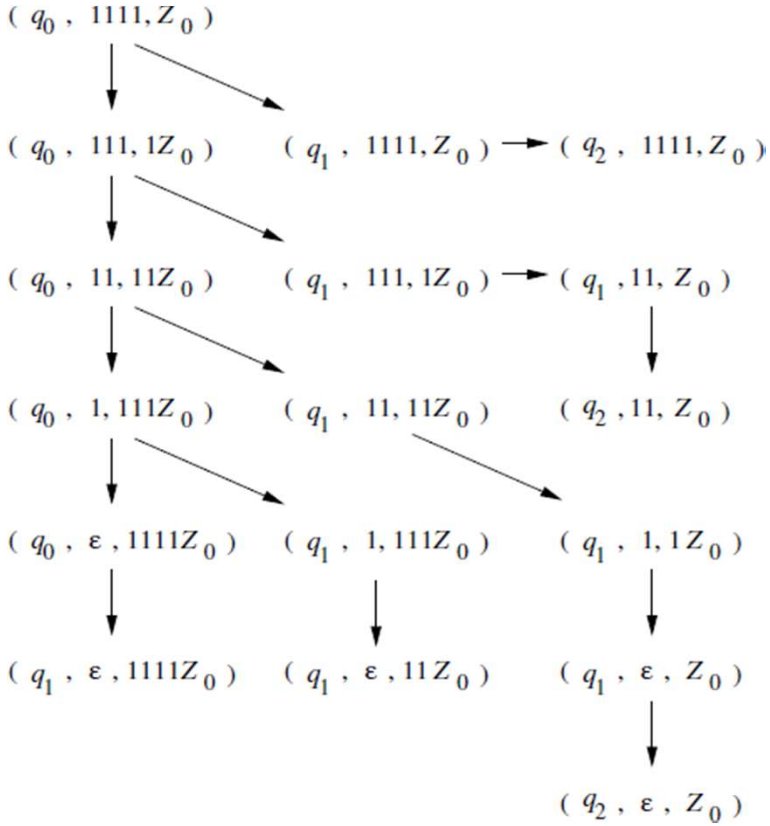


Arrows represent move ⊢ relation.



PDA Move - Example





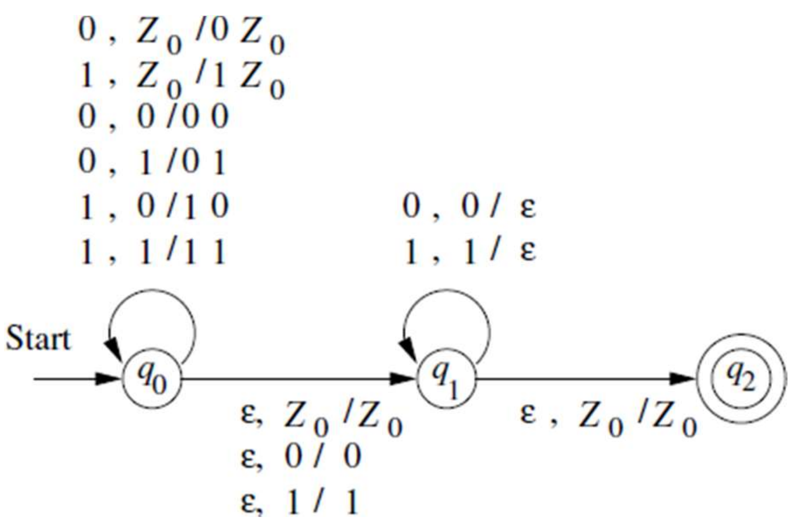
- A sequence of moves (a **computation**):

$(q_0,1111,Z_0) \vdash (q_0,111,1Z_0) \vdash (q_0,11,11Z_0) \vdash (q_1,11,11Z_0) \vdash (q_1,1,1Z_0) \vdash (q_1,\varepsilon,Z_0) \vdash (q_2,\varepsilon,Z_0)$

- **Thus, $(q_0,1111,Z_0) \vdash^* (q_2,\varepsilon,Z_0)$**

PDA Move - Example

- On input 01 the PDA has the following **computation sequences**:



$$\begin{array}{l}
(q_0,01,Z_0) \to (q_0,1,0Z_0) \\
(q_0,01,Z_0) \to (q_1,01,Z_0) \to (q_2,01,Z_0) \\
(q_0,1,0Z_0) \to (q_0,\varepsilon,10Z_0) \to (q_1,\varepsilon,10Z_0) \\
(q_0,1,0Z_0) \to (q_1,1,0Z_0)
\end{array}$$

- All computations end with an ID whose state is NOT a final state or the input string is NOT consumed. Thus, 01 is NOT accepted by this PDA.
  - $(q_1,\varepsilon,10Z_0)$ q_1 is NOT a final state, and q_1 does not have any move.
  - $(q_2,01,Z_0)$ the input (01) is NOT consumed, and q_2 does not have any move.