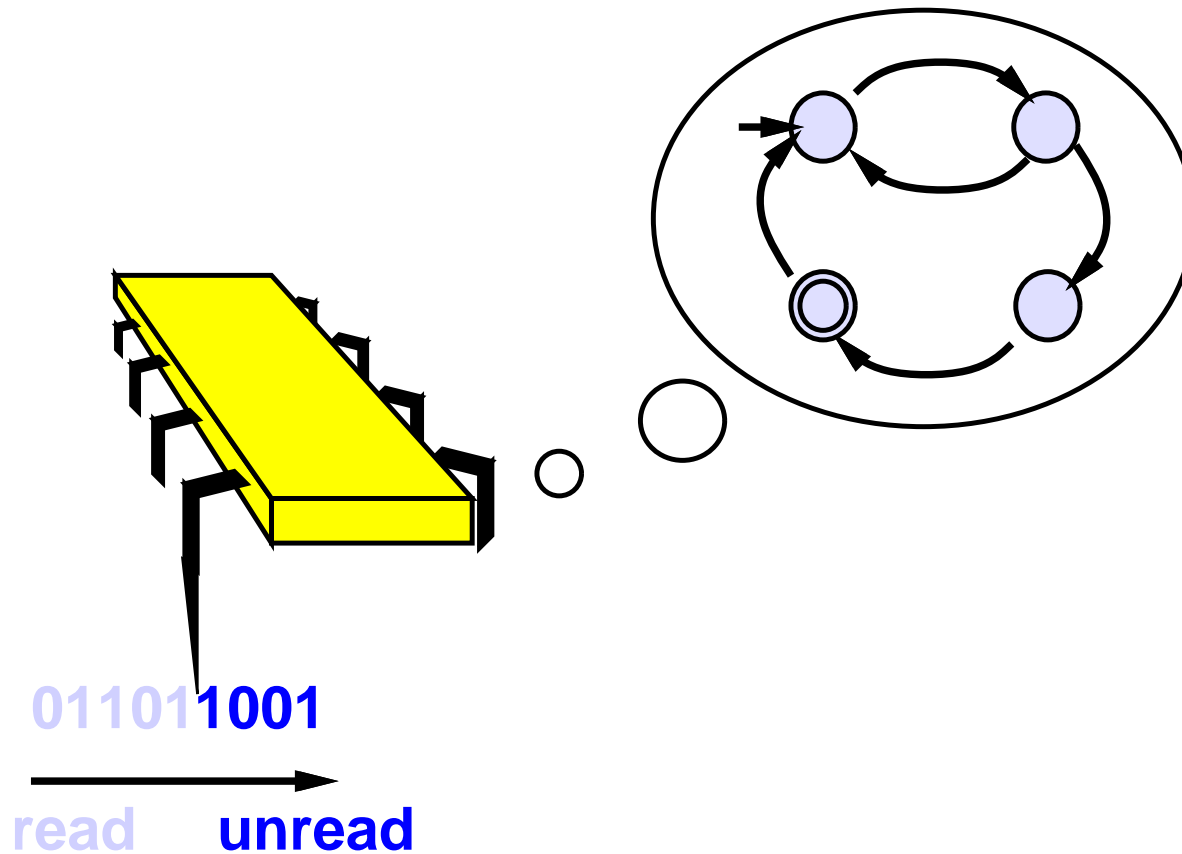
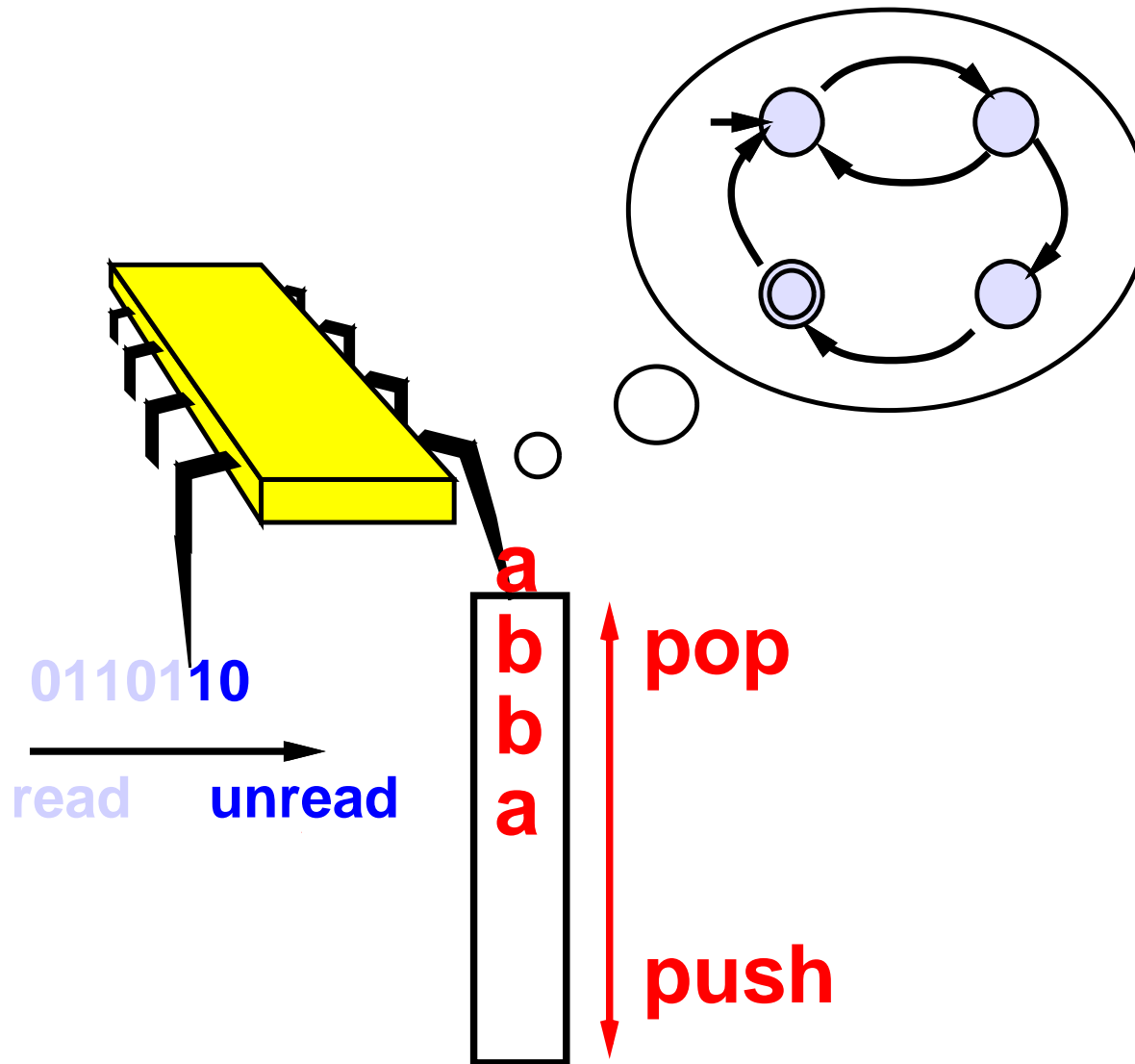


# A Finite Automaton



# A Pushdown Automaton



# A Pushdown Automaton

- can **push** symbols onto the stack

# A Pushdown Automaton

- can **push** symbols onto the stack
- can **pop** them (read them back) later

# A Pushdown Automaton

- can **push** symbols onto the stack
- can **pop** them (read them back) later
- stack can grow **unboundedly**

# A Pushdown Automaton

- can **push** symbols onto the stack
- can **pop** them (read them back) later
- stack can grow **unboundedly**
- yet at any moment stack has **finite** size.

# An Example

Recall that the language  $\{0^n 1^n \mid n \geq 0\}$  is not regular.  
Consider the following PDA:

- read input symbols

# An Example

Recall that the language  $\{0^n 1^n \mid n \geq 0\}$  is not regular.  
Consider the following PDA:

- read input symbols
- for each 0, push it on the stack

# An Example

Recall that the language  $\{0^n 1^n \mid n \geq 0\}$  is not regular.  
Consider the following PDA:

- read input symbols
- for each 0, push it on the stack
- as soon as a 1 is seen, pop a 0 for each 1 read

# An Example

Recall that the language  $\{0^n 1^n \mid n \geq 0\}$  is not regular.  
Consider the following PDA:

- read input symbols
- for each 0, push it on the stack
- as soon as a 1 is seen, pop a 0 for each 1 read
- **accept** if stack is **empty** when **last symbol read**

# An Example

Recall that the language  $\{0^n 1^n \mid n \geq 0\}$  is not regular.  
Consider the following PDA:

- read input symbols
- for each 0, push it on the stack
- as soon as a 1 is seen, pop a 0 for each 1 read
- **accept** if stack is **empty** when **last symbol read**
- **reject** if

# An Example

Recall that the language  $\{0^n 1^n \mid n \geq 0\}$  is not regular.  
Consider the following PDA:

- read input symbols
- for each 0, push it on the stack
- as soon as a 1 is seen, pop a 0 for each 1 read
- **accept** if stack is **empty** when last symbol read
- **reject** if
  - stack is **non-empty** when last symbol read

# An Example

Recall that the language  $\{0^n 1^n \mid n \geq 0\}$  is not regular. Consider the following PDA:

- read input symbols
- for each 0, push it on the stack
- as soon as a 1 is seen, pop a 0 for each 1 read
- **accept** if stack is **empty** when last symbol read
- **reject** if
  - stack is **non-empty** when last symbol read
  - stack is **empty** but **input symbol(s)** still exist,

# An Example

Recall that the language  $\{0^n 1^n \mid n \geq 0\}$  is not regular. Consider the following PDA:

- read input symbols
- for each 0, push it on the stack
- as soon as a 1 is seen, pop a 0 for each 1 read
- **accept** if stack is **empty** when last symbol read
- **reject** if
  - stack is **non-empty** when last symbol read
  - stack is **empty** but **input symbol(s)** still exist,
  - 0 is read after 1.

# On PDA vs. Finite Automata

- Nondeterminism

# On PDA vs. Finite Automata

- Nondeterminism
  - PDA may be deterministic or non-deterministic.

# On PDA vs. Finite Automata

- Nondeterminism
  - PDA may be deterministic or non-deterministic.
  - Unlike finite automata, non-determinism adds power.

# On PDA vs. Finite Automata

- Nondeterminism
  - PDA may be deterministic or non-deterministic.
  - Unlike finite automata, non-determinism adds power.
  - There are some languages accepted **only** by **non-deterministic** PDAs.

# On PDA vs. Finite Automata

- **Nondeterminism**
  - PDA may be deterministic or non-deterministic.
  - Unlike finite automata, non-determinism adds power.
  - There are some languages accepted **only** by **non-deterministic** PDAs.
- **Transition function  $\delta$**  looks different than DFA or NFA cases, reflecting **stack** functionality.

# The Transition Function

Denote input alphabet by  $\Sigma$  and stack alphabet by  $\Gamma$ .

- the **domain** of the transition function  $\delta$  is

# The Transition Function

Denote input alphabet by  $\Sigma$  and stack alphabet by  $\Gamma$ .

- the **domain** of the transition function  $\delta$  is
  - current state:  $Q$

# The Transition Function

Denote input alphabet by  $\Sigma$  and stack alphabet by  $\Gamma$ .

- the **domain** of the transition function  $\delta$  is
  - current state:  $Q$
  - next input symbol, if any:  $\Sigma_\epsilon (= \Sigma \cup \{\epsilon\})$

# The Transition Function

Denote input alphabet by  $\Sigma$  and stack alphabet by  $\Gamma$ .

- the **domain** of the transition function  $\delta$  is
  - current state:  $Q$
  - next input symbol, if any:  $\Sigma_\epsilon (= \Sigma \cup \{\epsilon\})$
  - stack symbol popped, if any:  $\Gamma_\epsilon$

# The Transition Function

Denote input alphabet by  $\Sigma$  and stack alphabet by  $\Gamma$ .

- the **domain** of the transition function  $\delta$  is
  - current state:  $Q$
  - next input symbol, if any:  $\Sigma_\epsilon (= \Sigma \cup \{\epsilon\})$
  - stack symbol popped, if any:  $\Gamma_\epsilon$
- and its **range** is

# The Transition Function

Denote input alphabet by  $\Sigma$  and stack alphabet by  $\Gamma$ .

- the **domain** of the transition function  $\delta$  is
  - current state:  $Q$
  - next input symbol, if any:  $\Sigma_\epsilon (= \Sigma \cup \{\epsilon\})$
  - stack symbol popped, if any:  $\Gamma_\epsilon$
- and its **range** is
  - new state:  $Q$

# The Transition Function

Denote input alphabet by  $\Sigma$  and stack alphabet by  $\Gamma$ .

- the **domain** of the transition function  $\delta$  is
  - current state:  $Q$
  - next input symbol, if any:  $\Sigma_\epsilon (= \Sigma \cup \{\epsilon\})$
  - stack symbol popped, if any:  $\Gamma_\epsilon$
- and its **range** is
  - new state:  $Q$
  - stack symbol pushed, if any:  $\Gamma_\epsilon$

# The Transition Function

Denote input alphabet by  $\Sigma$  and stack alphabet by  $\Gamma$ .

- the **domain** of the transition function  $\delta$  is
  - current state:  $Q$
  - next input symbol, if any:  $\Sigma_\epsilon (= \Sigma \cup \{\epsilon\})$
  - stack symbol popped, if any:  $\Gamma_\epsilon$
- and its **range** is
  - new state:  $Q$
  - stack symbol pushed, if any:  $\Gamma_\epsilon$
  - non-determinism:  $\mathcal{P}(\dots)$

# The Transition Function

Denote input alphabet by  $\Sigma$  and stack alphabet by  $\Gamma$ .

- the **domain** of the transition function  $\delta$  is
  - current state:  $Q$
  - next input symbol, if any:  $\Sigma_\epsilon (= \Sigma \cup \{\epsilon\})$
  - stack symbol popped, if any:  $\Gamma_\epsilon$
- and its **range** is
  - new state:  $Q$
  - stack symbol pushed, if any:  $\Gamma_\epsilon$
  - non-determinism:  $\mathcal{P}(\dots)$
- $\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \rightarrow \mathcal{P}(Q \times \Gamma_\epsilon)$

# Formal Definitions

A pushdown automaton (PDA) is a 6-tuple  $(Q, \Sigma, \Gamma, \delta, q_0, F)$ , where

- $Q$  is a finite set called the states,

# Formal Definitions

A pushdown automaton (PDA) is a 6-tuple  $(Q, \Sigma, \Gamma, \delta, q_0, F)$ , where

- $Q$  is a finite set called the **states**,
- $\Sigma$  is a finite set called the **input alphabet**,

# Formal Definitions

A pushdown automaton (PDA) is a 6-tuple  $(Q, \Sigma, \Gamma, \delta, q_0, F)$ , where

- $Q$  is a finite set called the **states**,
- $\Sigma$  is a finite set called the **input alphabet**,
- $\Gamma$  is a finite set called the **stack alphabet**,

# Formal Definitions

A **pushdown automaton** (PDA) is a 6-tuple  $(Q, \Sigma, \Gamma, \delta, q_0, F)$ , where

- $Q$  is a finite set called the **states**,
- $\Sigma$  is a finite set called the **input alphabet**,
- $\Gamma$  is a finite set called the **stack alphabet**,
- $\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \rightarrow \mathcal{P}(Q \times \Gamma_\epsilon)$  is the **transition function**,

# Formal Definitions

A **pushdown automaton** (PDA) is a 6-tuple  $(Q, \Sigma, \Gamma, \delta, q_0, F)$ , where

- $Q$  is a finite set called the **states**,
- $\Sigma$  is a finite set called the **input alphabet**,
- $\Gamma$  is a finite set called the **stack alphabet**,
- $\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \rightarrow \mathcal{P}(Q \times \Gamma_\epsilon)$  is the **transition function**,
- $q_0 \in Q$  is the **start state**, and

# Formal Definitions

A **pushdown automaton** (PDA) is a 6-tuple  $(Q, \Sigma, \Gamma, \delta, q_0, F)$ , where

- $Q$  is a finite set called the **states**,
- $\Sigma$  is a finite set called the **input alphabet**,
- $\Gamma$  is a finite set called the **stack alphabet**,
- $\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \rightarrow \mathcal{P}(Q \times \Gamma_\epsilon)$  is the **transition function**,
- $q_0 \in Q$  is the **start state**, and
- $F \subseteq Q$  is the set of **accept states**.

# Conventions

- Question: When is the stack empty?

# Conventions

- **Question:** When is the stack empty?
- start by pushing \$ onto stack

# Conventions

- **Question:** When is the stack empty?
  - start by pushing \$ onto stack
  - when you see it again, stack is empty.

# Conventions

- **Question:** When is the stack empty?
  - start by pushing \$ onto stack
  - when you see it again, stack is empty.
- **Question:** When is input string exhausted?

# Conventions

- **Question:** When is the stack empty?
  - start by pushing \$ onto stack
  - when you see it again, stack is empty.
- **Question:** When is input string exhausted?
  - doesn't matter – PDA need not know this explicitly

# Conventions

- **Question:** When is the stack empty?
  - start by pushing \$ onto stack
  - when you see it again, stack is empty.
- **Question:** When is input string exhausted?
  - doesn't matter – PDA need not know this explicitly
  - accepting state accepts only if inputs exhausted!

# Notation

- Transition  $a, b \rightarrow c$  means

# Notation

- Transition  $a, b \rightarrow c$  means
  - read  $a$  from input

# Notation

- Transition  $a, b \rightarrow c$  means
  - read  $a$  from input
  - pop  $b$  from stack

# Notation

- Transition  $a, b \rightarrow c$  means
  - read  $a$  from input
  - pop  $b$  from stack
  - push  $c$  onto stack

# Notation

- Transition  $a, b \rightarrow c$  means
  - read  $a$  from input
  - pop  $b$  from stack
  - push  $c$  onto stack
- Meaning of  $\varepsilon$  transitions:

# Notation

- Transition  $a, b \rightarrow c$  means
  - read  $a$  from input
  - pop  $b$  from stack
  - push  $c$  onto stack
- Meaning of  $\varepsilon$  transitions:
  - if  $a = \varepsilon$ , don't read inputs

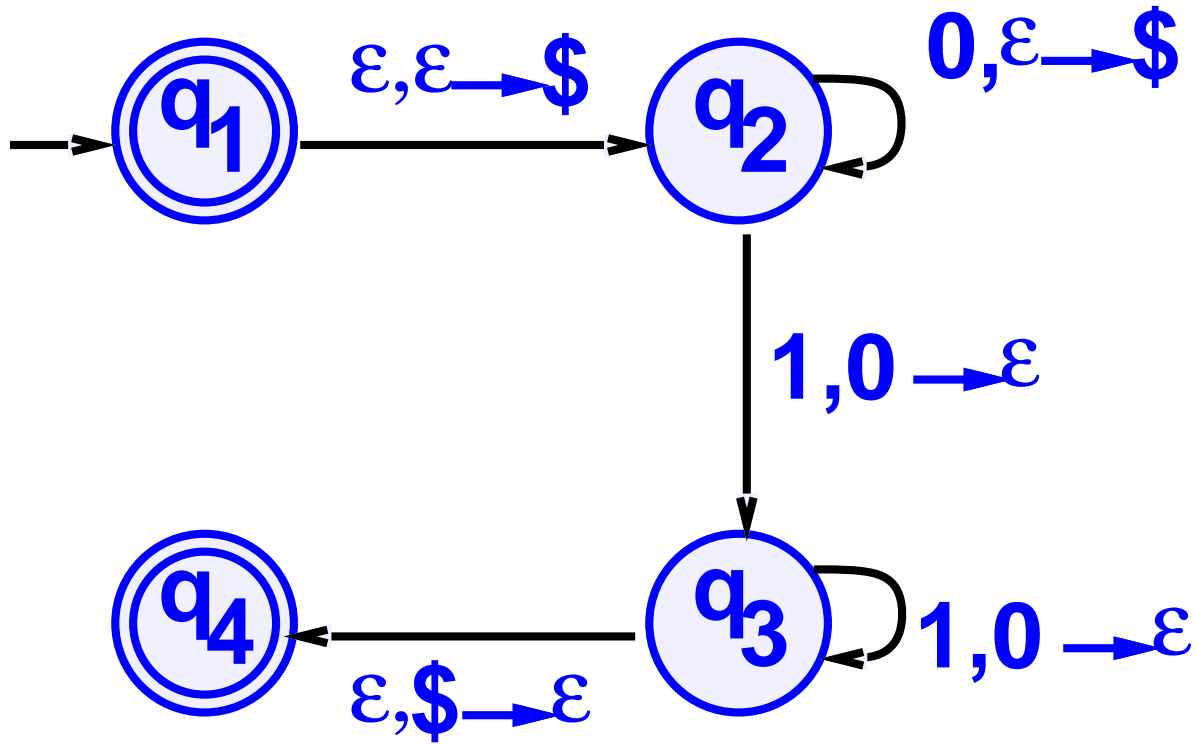
# Notation

- Transition  $a, b \rightarrow c$  means
  - read  $a$  from input
  - pop  $b$  from stack
  - push  $c$  onto stack
- Meaning of  $\varepsilon$  transitions:
  - if  $a = \varepsilon$ , don't read inputs
  - if  $b = \varepsilon$ , don't pop any symbols

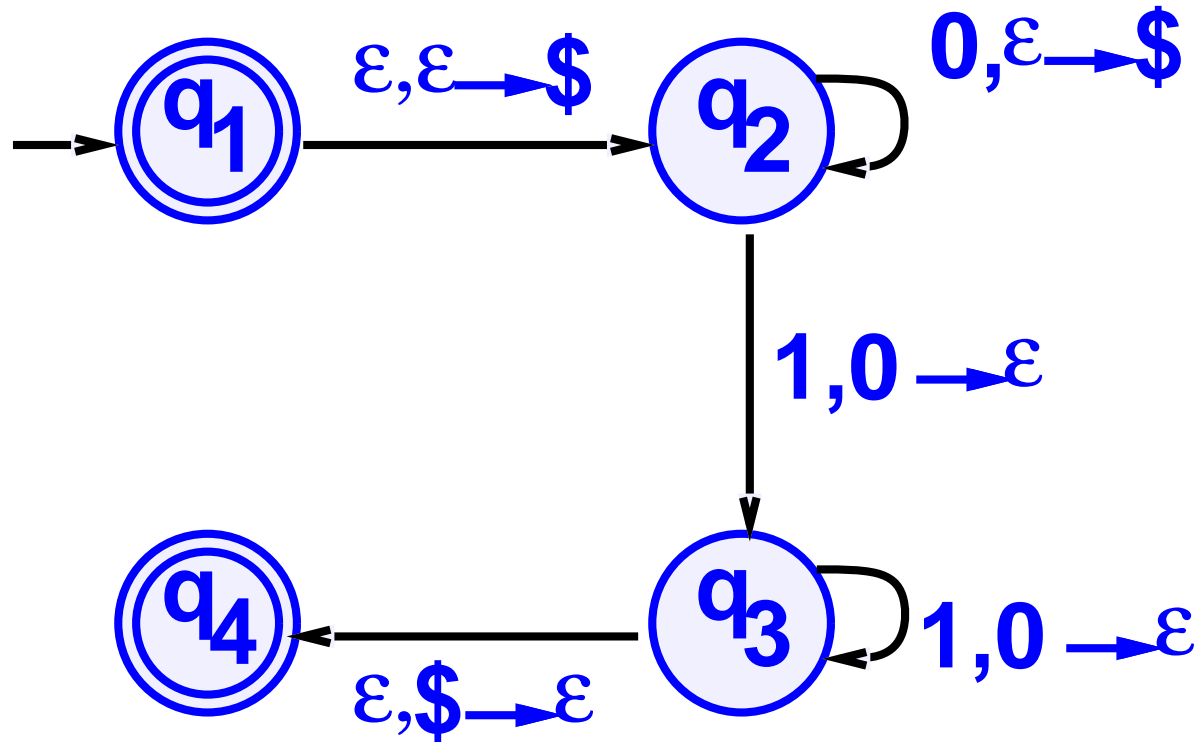
# Notation

- Transition  $a, b \rightarrow c$  means
  - read  $a$  from input
  - pop  $b$  from stack
  - push  $c$  onto stack
- Meaning of  $\varepsilon$  transitions:
  - if  $a = \varepsilon$ , don't read inputs
  - if  $b = \varepsilon$ , don't pop any symbols
  - if  $c = \varepsilon$ , don't push any symbols

# Example

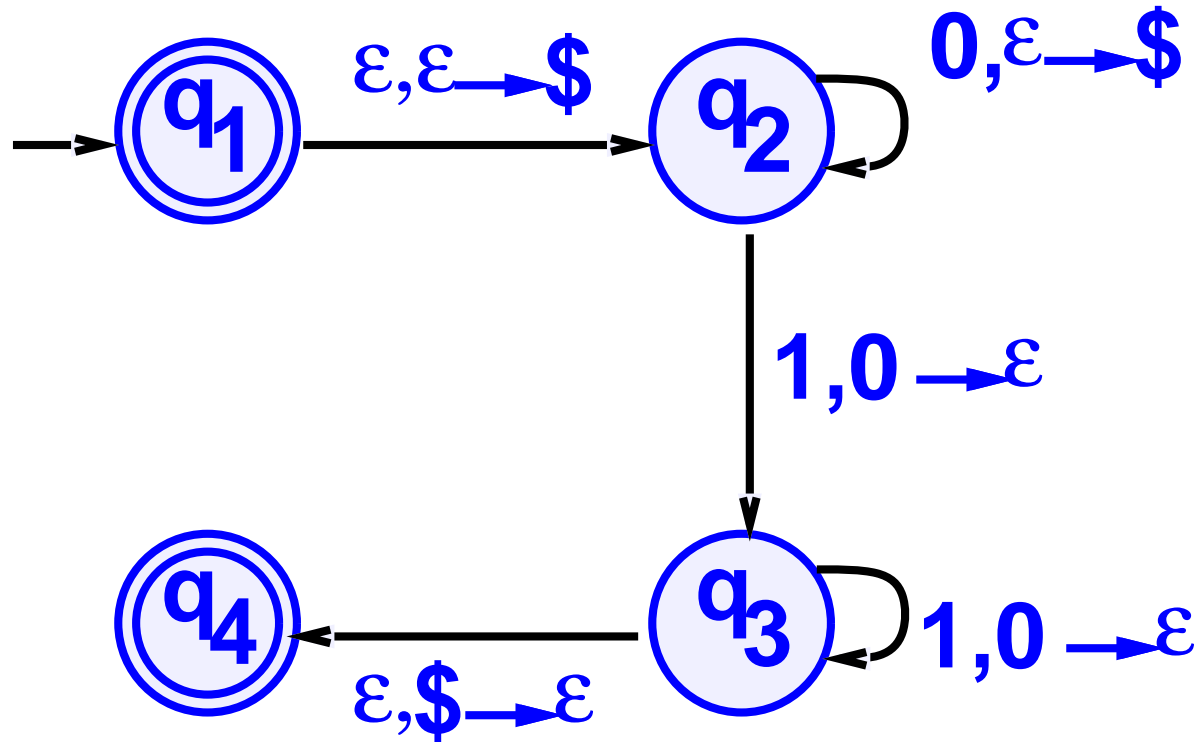


# Example



What does this PDA accept?

# Example



What does this PDA accept?

$\{0^n 1^n \mid n \geq 1\}$ .

# Another Example

A PDA that accepts

$$\{a^i b^j c^k \mid i, j, k > 0 \text{ and } i = j \text{ or } i = k\}$$

Informally:

- read and push  $a$ 's

# Another Example

A PDA that accepts

$$\{a^i b^j c^k \mid i, j, k > 0 \text{ and } i = j \text{ or } i = k\}$$

Informally:

- read and push  $a$ 's
- either pop and match with  $b$ 's

# Another Example

A PDA that accepts

$$\{a^i b^j c^k \mid i, j, k > 0 \text{ and } i = j \text{ or } i = k\}$$

Informally:

- read and push  $a$ 's
- either pop and match with  $b$ 's
- or else pop and match with  $c$ 's

# Another Example

A PDA that accepts

$$\{a^i b^j c^k \mid i, j, k > 0 \text{ and } i = j \text{ or } i = k\}$$

Informally:

- read and push  $a$ 's
- either pop and match with  $b$ 's
- or else pop and match with  $c$ 's
- non-deterministic choice!

# Another Example

A PDA that accepts

$$\{a^i b^j c^k \mid i, j, k > 0 \text{ and } i = j \text{ or } i = k\}$$

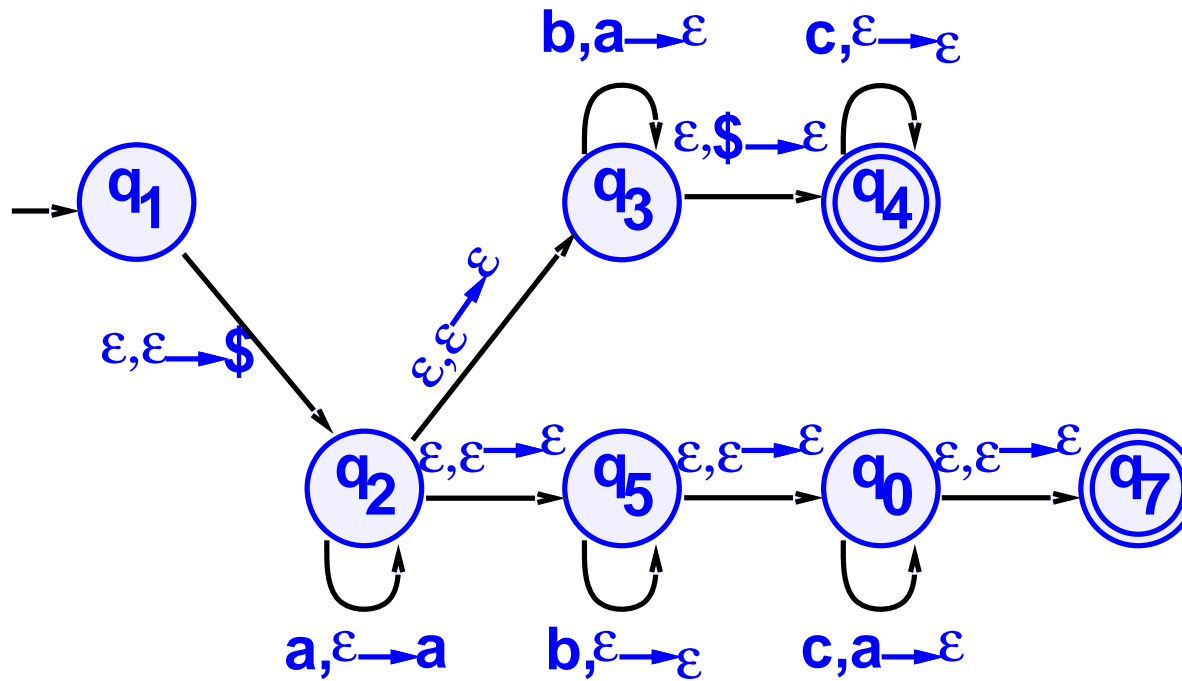
Informally:

- read and push  $a$ 's
- either pop and match with  $b$ 's
- or else pop and match with  $c$ 's
- non-deterministic choice!

**Note:** non-determinism **essential** here!

Unlike finite automata, non-determinism does add power

# Another Example



This PDA accepts

$$\{a^i b^j c^k \mid i, j, k > 0 \text{ and } i = j \text{ or } i = k\}$$

# Yet Another Example

A **palindrome** is a string  $w$  satisfying  $w = w^R$ .

- “Madam I’m Adam”

# Yet Another Example

A **palindrome** is a string  $w$  satisfying  $w = w^R$ .

- “Madam I’m Adam”
- “Dennis and Edna sinned”

# Yet Another Example

A **palindrome** is a string  $w$  satisfying  $w = w^R$ .

- “Madam I’m Adam”
- “Dennis and Edna sinned”
- “Red rum, sir, is murder”

# Yet Another Example

A **palindrome** is a string  $w$  satisfying  $w = w^R$ .

- “Madam I’m Adam”
- “Dennis and Edna sinned”
- “Red rum, sir, is murder”
- “Able was I ere I saw Elba”

# Yet Another Example

A **palindrome** is a string  $w$  satisfying  $w = w^R$ .

- “Madam I’m Adam”
- “Dennis and Edna sinned”
- “Red rum, sir, is murder”
- “Able was I ere I saw Elba”
- “In girum imus nocte et consumimur igni”

# Yet Another Example

A **palindrome** is a string  $w$  satisfying  $w = w^R$ .

- “Madam I’m Adam”
- “Dennis and Edna sinned”
- “Red rum, sir, is murder”
- “Able was I ere I saw Elba”
- “In girum imus nocte et consumimur igni”
- “*νιψον ανομηματα μη μοναν οψιν*”

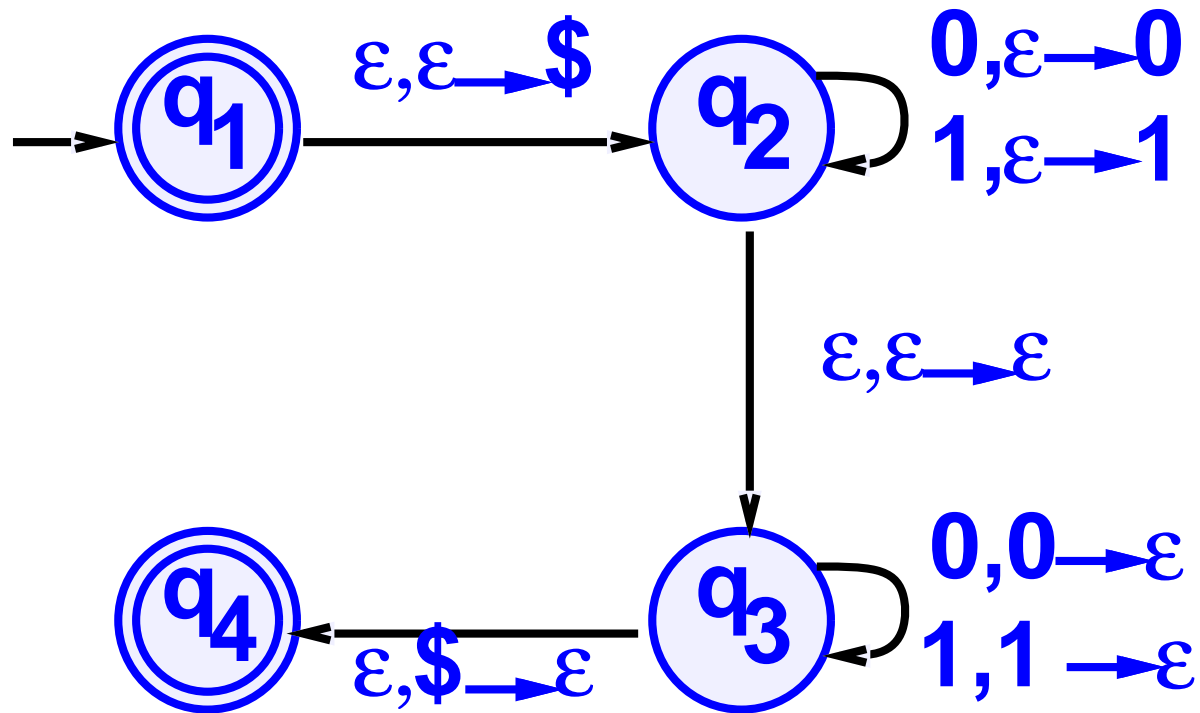
# Yet Another Example

A **palindrome** is a string  $w$  satisfying  $w = w^R$ .

- “Madam I’m Adam”
- “Dennis and Edna sinned”
- “Red rum, sir, is murder”
- “Able was I ere I saw Elba”
- “In girum imus nocte et consumimur igni”
- “*νιψον ανομηματα μη μοναν οψιν*”

Palindromes also appear in nature. For example as DNA sites (strings over  $\{A, C, T, G\}$ ) being cut by restriction enzymes.

## Yet Another Example



This PDA accepts binary palindromes of *even length*.

# Equivalence Theorem

**Theorem:** A language is context free if and only if some pushdown automata accepts it.

# Equivalence Theorem

**Theorem:** A language is context free if and only if some pushdown automata accepts it.

This time, the proofs of both the “if” part and the “only if” part are interesting.

## If Part

**Theorem:** If a language is context free, then some pushdown automaton accepts it.

- Let  $A$  be a context-free language.

## If Part

**Theorem:** If a language is context free, then some pushdown automaton accepts it.

- Let  $A$  be a context-free language.
- By definition,  $A$  has a context-free grammar  $G$  generating it.

## If Part

**Theorem:** If a language is context free, then some pushdown automaton accepts it.

- Let  $A$  be a context-free language.
- By definition,  $A$  has a context-free grammar  $G$  generating it.
- On input  $w$ , the PDA  $P$  should figure out if there is a derivation of  $w$  using  $G$ .

## If Part

**Theorem:** If a language is context free, then some pushdown automaton accepts it.

- Let  $A$  be a context-free language.
- By definition,  $A$  has a context-free grammar  $G$  generating it.
- On input  $w$ , the PDA  $P$  should figure out if there is a derivation of  $w$  using  $G$ .

**Question:** How does  $P$  figure out which substitution to make?

## If Part

**Theorem:** If a language is context free, then some pushdown automaton accepts it.

- Let  $A$  be a context-free language.
- By definition,  $A$  has a context-free grammar  $G$  generating it.
- On input  $w$ , the PDA  $P$  should figure out if there is a derivation of  $w$  using  $G$ .

**Question:** How does  $P$  figure out which substitution to make?

**Answer:** It guesses.

# CFL Implies PDA

Informally:

- $P$  pushes start variable  $S$  on stack

# CFL Implies PDA

Informally:

- $P$  pushes start variable  $S$  on stack
- keeps making substitutions, storing **intermediate strings**

# CFL Implies PDA

Informally:

- $P$  pushes start variable  $S$  on stack
- keeps making substitutions, storing **intermediate strings**
- when only terminals remain ...

# CFL Implies PDA

Informally:

- $P$  pushes start variable  $S$  on stack
- keeps making substitutions, storing **intermediate strings**
- when only terminals remain ...
- tests whether derived string equals input

# CFL Implies PDA

Where do we keep the **intermediate string**?

- Can't put it all on the stack

# CFL Implies PDA

Where do we keep the **intermediate string**?

- Can't put it all on the stack
- Keep on stack only **symbols after first variable**

# CFL Implies PDA

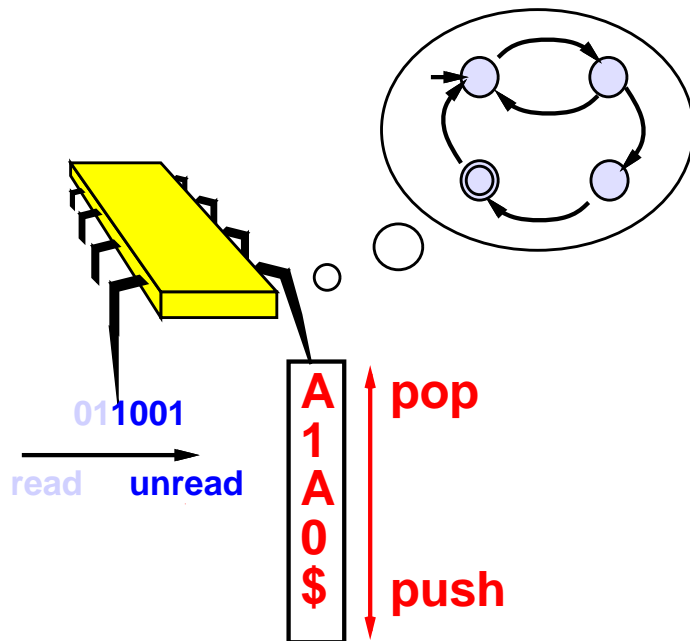
Where do we keep the **intermediate string**?

- Can't put it all on the stack
- Keep on stack only **symbols after first variable**
- Terminal symbols **before first variable** matched immediately to input string symbols.

# CFL Implies PDA

Where do we keep the **intermediate string**?

- Can't put it all on the stack
- Keep on stack only **symbols after first variable**
- Terminal symbols **before first variable** matched immediately to input string symbols.



**intermediate string:** 0**1A1A**0

# CFL Implies PDA

Informal description:

- push  $S\$$  on stack

# CFL Implies PDA

Informal description:

- push  $S\$\$  on stack
- if top of stack is a variable  $A$ ,  
non-deterministically select rule and substitute.

# CFL Implies PDA

Informal description:

- push  $S\$$  on stack
- if top of stack is a variable  $A$ , non-deterministically select rule and substitute.
- if top of stack is terminal  $a$  read next input and compare. If they differ, reject on this branch of the nondeterminism.

# CFL Implies PDA

Informal description:

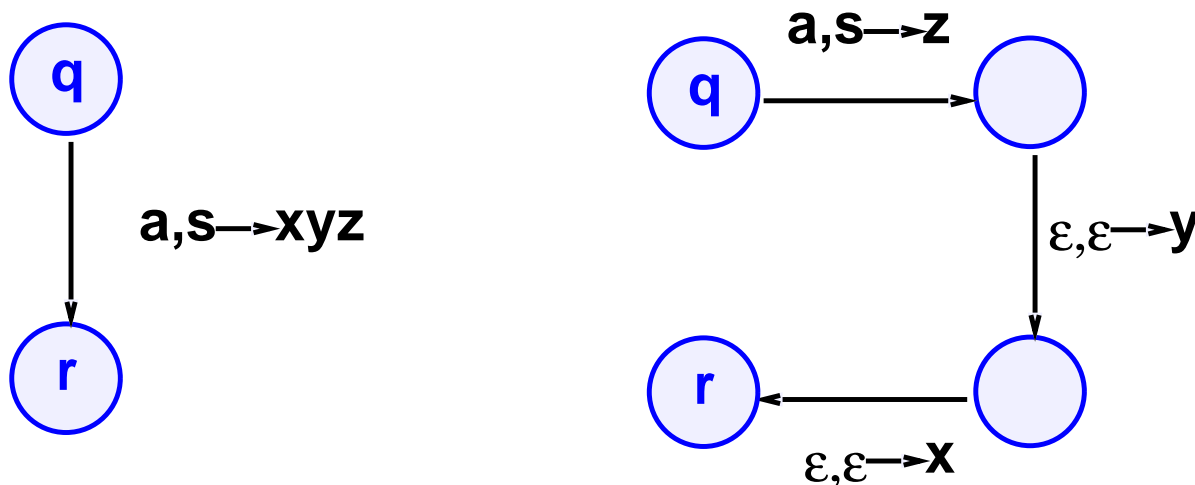
- push  $S\$$  on stack
- if top of stack is a variable  $A$ , non-deterministically select rule and substitute.
- if top of stack is terminal  $a$  read next input and compare. If they differ, reject on this branch of the nondeterminism.
- if top of stack is  $\$$ , enter accept state (here we accept only if input has all been read, as accept state will have no exits labeled with input symbols!).

# CFL Implies PDA

Need shorthand to push entire string onto stack (in this example the pushed string is  $w = xyz$ ).

$$(r, w) \in \delta(q, a, s)$$

Easy to do by introducing intermediate states.



# CFL Implies PDA

States of  $P$  are

- start state,  $q_s$

# CFL Implies PDA

States of  $P$  are

- start state,  $q_s$
- accept state,  $q_a$

# CFL Implies PDA

States of  $P$  are

- start state,  $q_s$
- accept state,  $q_a$
- loop state,  $q_\ell$

# CFL Implies PDA

States of  $P$  are

- start state,  $q_s$
- accept state,  $q_a$
- loop state,  $q_\ell$
- $E$  states, shorthand for pushing entire strings

# Transition Function

- Initialize stack –  $\delta(q_s, \varepsilon, \varepsilon) = \{q_l, S\}$

# Transition Function

- Initialize stack –  $\delta(q_s, \varepsilon, \varepsilon) = \{q_\ell, S\$\}$
- Top of stack is variable –  
 $\delta(q_\ell, \varepsilon, A) = \{(q_\ell, w) \mid \text{where } A \rightarrow w \text{ is a rule} \}$

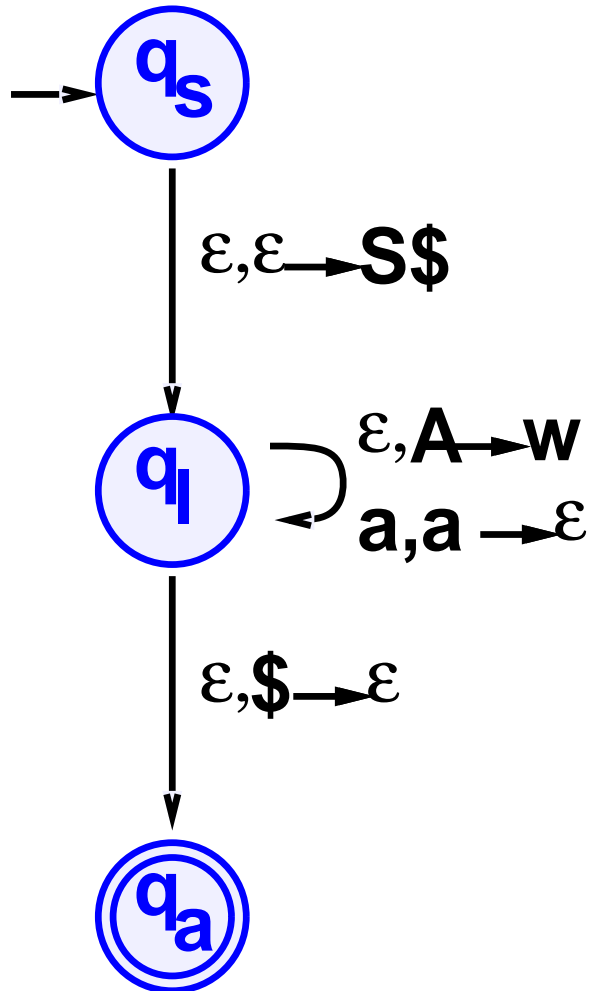
# Transition Function

- Initialize stack –  $\delta(q_s, \varepsilon, \varepsilon) = \{q_\ell, S\$\}$
- Top of stack is variable –  
 $\delta(q_\ell, \varepsilon, A) = \{(q_\ell, w) \mid \text{where } A \rightarrow w \text{ is a rule} \}$
- Top of stack is terminal –  $\delta(q_\ell, a, a) = \{(q_\ell, \varepsilon)\}$

# Transition Function

- Initialize stack –  $\delta(q_s, \varepsilon, \varepsilon) = \{q_\ell, S\}$
- Top of stack is variable –  
 $\delta(q_\ell, \varepsilon, A) = \{(q_\ell, w) \mid \text{where } A \rightarrow w \text{ is a rule}\}$
- Top of stack is terminal –  $\delta(q_\ell, a, a) = \{(q_\ell, \varepsilon)\}$
- End of Stack –  $\delta(q_\ell, \varepsilon, \$) = \{(q_a, \varepsilon)\}$

# Transition Function

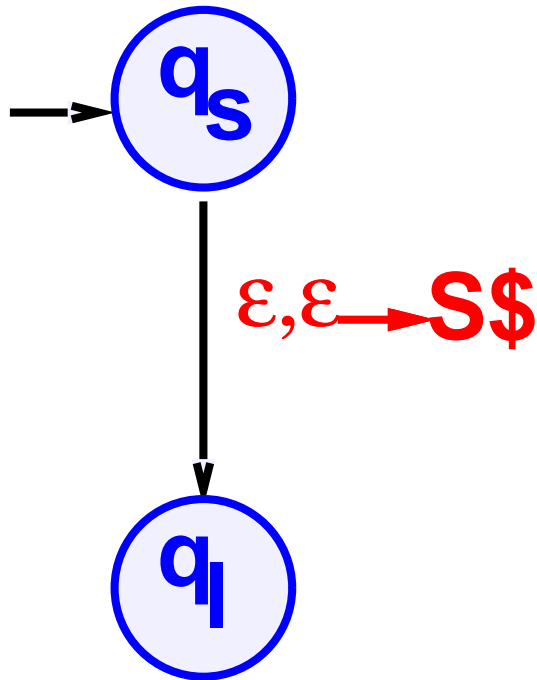


# Example

$$S \rightarrow aTb|b$$

$$T \rightarrow Ta|\varepsilon$$

Initialization:

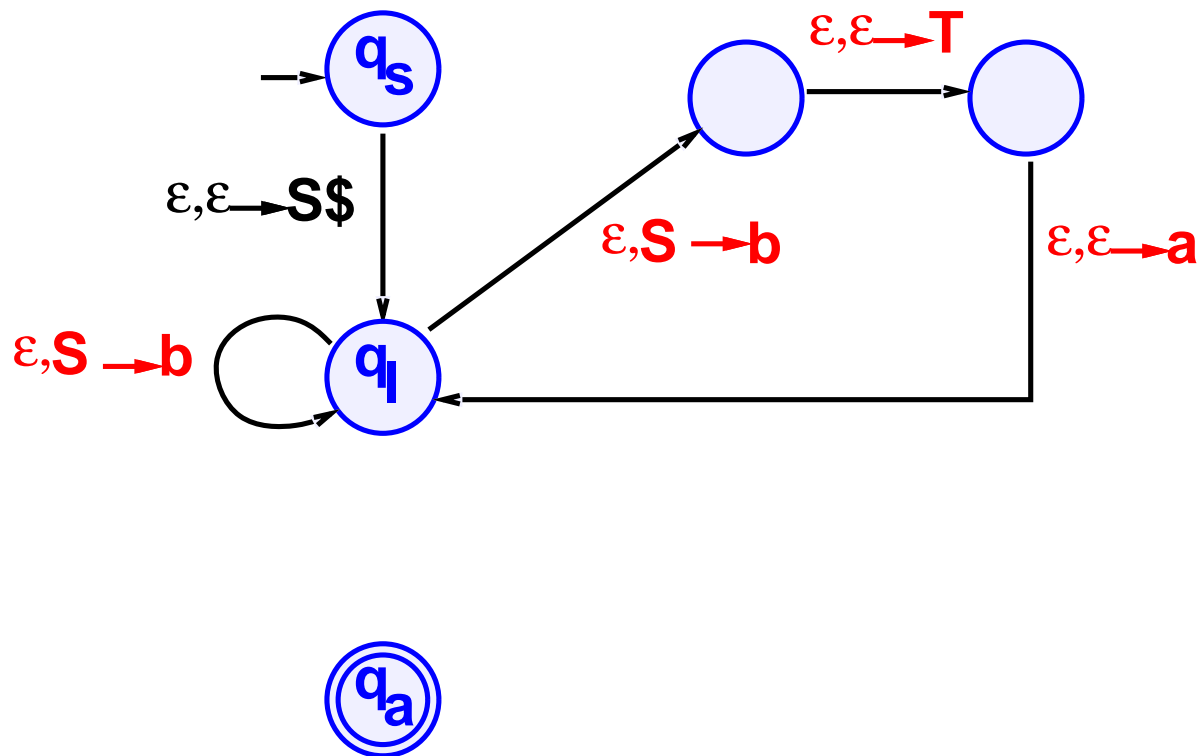


# Example

$$S \rightarrow aTb|b$$

$$T \rightarrow Ta|\varepsilon$$

## Rules for $S$

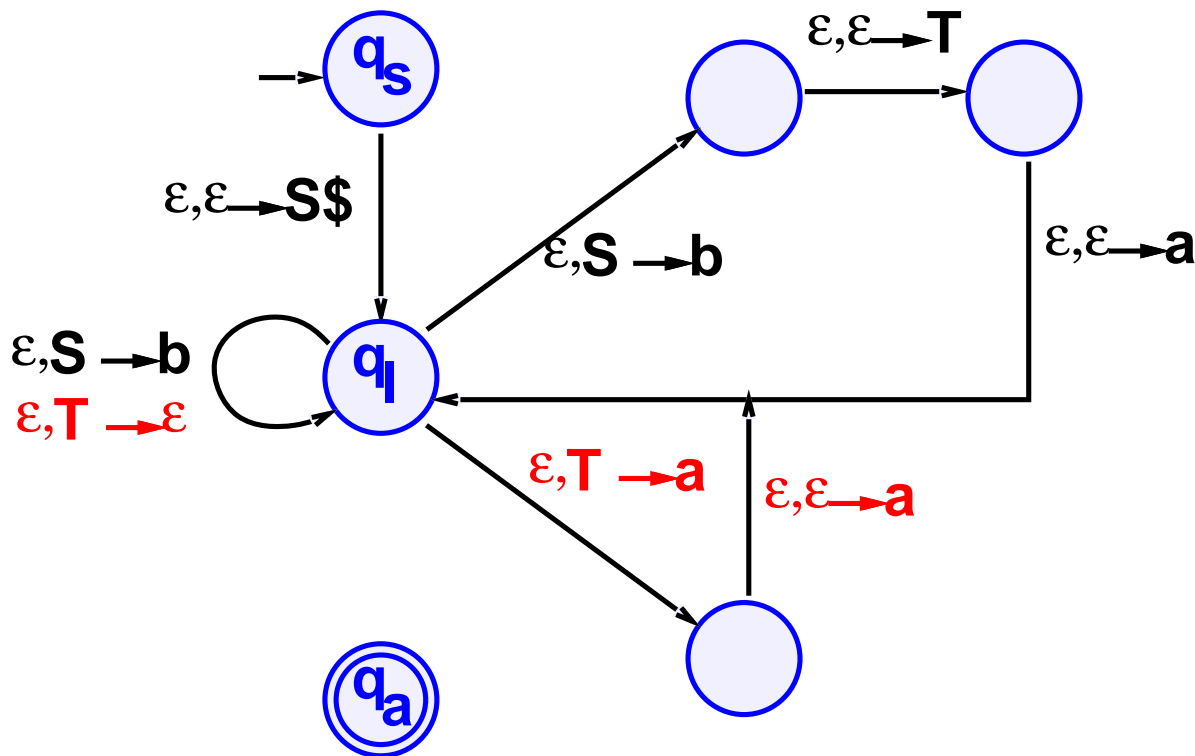


# Example

$$S \rightarrow aTb|b$$

$$T \rightarrow Ta|\varepsilon$$

## Rules for $T$

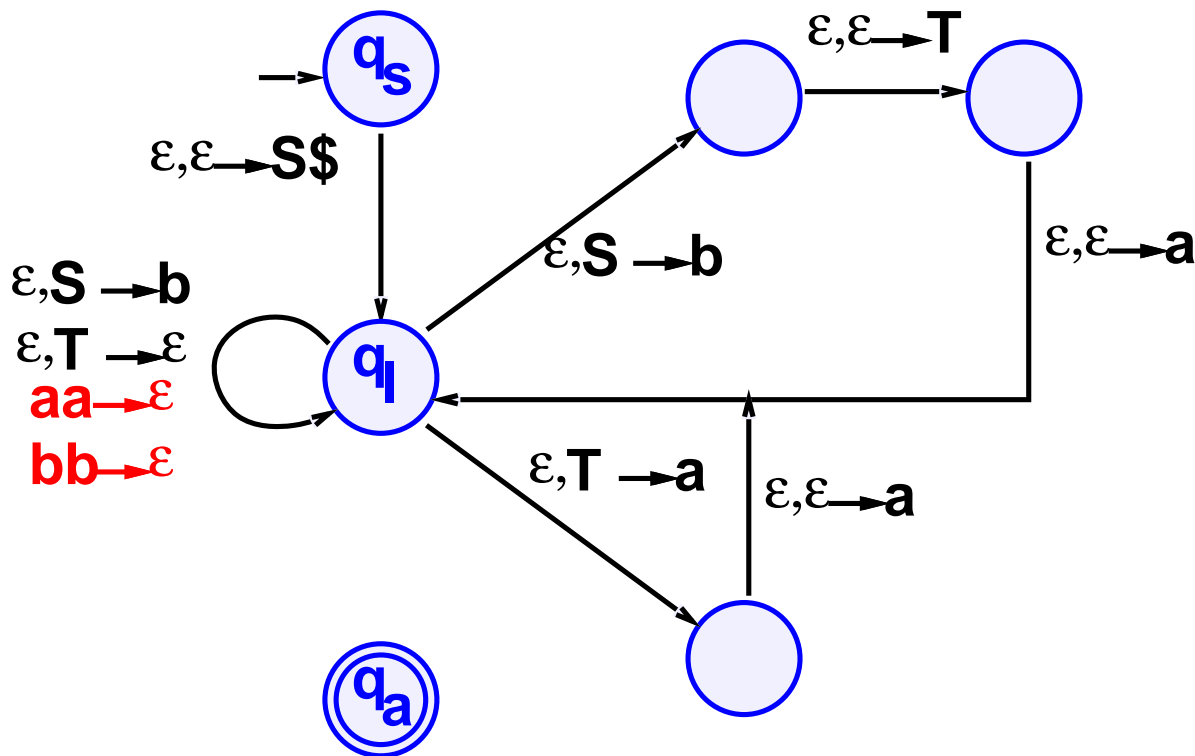


# Example

$$S \rightarrow aTb|b$$

$$T \rightarrow Ta|\varepsilon$$

## Rules for terminals



# Example

$$S \rightarrow aTb|b$$

$$T \rightarrow Ta|\varepsilon$$

Termination:

