Theory of Computation

CFG and PDA Equivalence

Manar Elkady, Ph.D.

Some of this lecture slides are taken from previous course of prof. Dr.Abdulhussein Mohsin Abdullah

- **CFG** and **PDA** are equivalent in power: both specify context-free languages

- We show here how to convert a CFG into a PDA that recognizes the language specified by the CFG and vice versa

- **CFG** can **specify** a programming language and the equivalent **PDA** its **compiler**

Theorem:

A language is context free if and only if some PDA recognizes it.

Proved in two lemmas

1. If a language is context-free, then some **PDA** recognizes it.

2. If a **PDA** recognizes a language, then it is context-free.

CFG's are recognized by PDA's

Lemma:

If a language is context free, then some PDA recognizes it.

Proof idea:

Construct a **PDA** following **CFG** rules

Constructing the PDA

- $Q = \{q_{start}, q_{loop}, q_{accept}\} \cup E$, where E is the set of states used for replacement rules onto the stack

- Σ (the PDA alphabet) is the set of terminals in the CFG

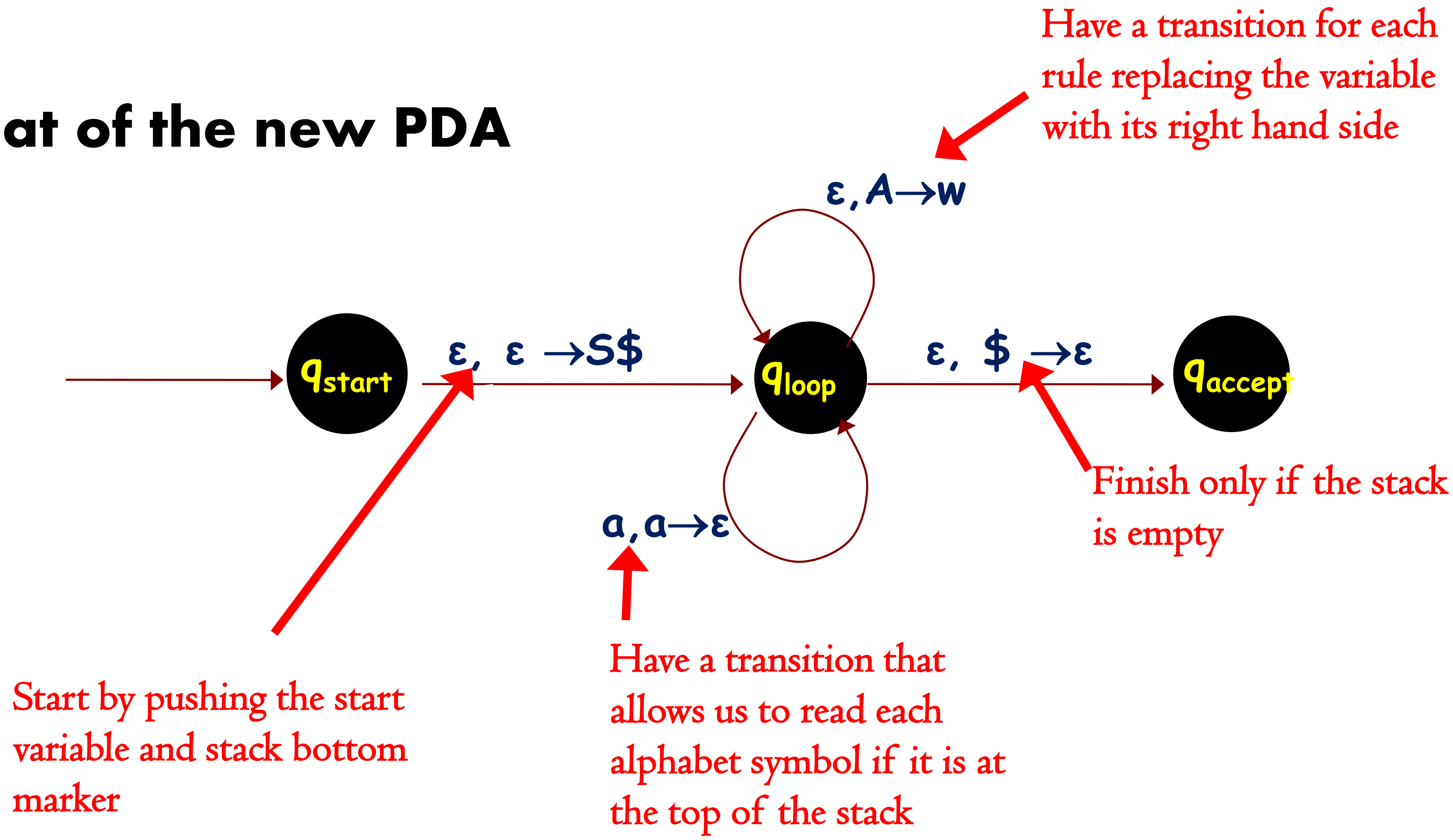
- Γ (the stack alphabet) is the union of the terminals and the variables and $\{\$\}$ (or some suitable placeholder)

Constructing the PDA

- δ is comprised of several rules
  1. $\delta(q_{start}, \varepsilon, \varepsilon) = (q_{loop}, \$)$
     $\delta(q_{start}, \varepsilon, \varepsilon) = (q_{loop}, S)$
     - Start with placeholder on the stack and with the start variable
  2. $\delta(q_{loop}, a, a) = (q_{loop}, \varepsilon)$ for every $a \in \Sigma$
     - Terminals may be read off the top of the stack
  3. $\delta(q_{loop}, \varepsilon, A) = (q_{loop}, w)$ for every rule $A \to w$
     - Implement replacement rules
  4. $\delta(q_{loop}, \varepsilon, \$) = (q_{accept}, \varepsilon)$
     - Accept when the stack is empty

CFG's are recognized by PDA's

Format of the new PDA



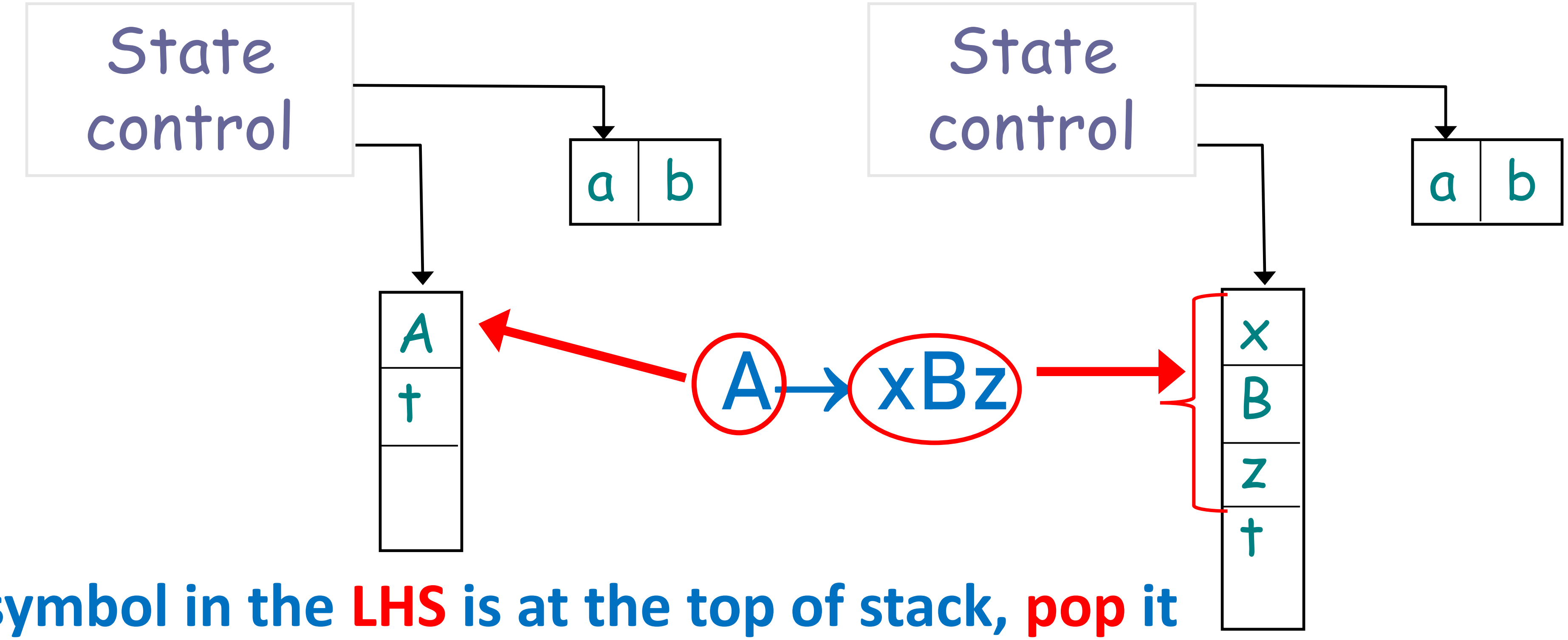
Constructing the PDA

- You can read any symbol in Σ when that symbol is at the top of the stack.

    - Transitions of the form $a,a \rightarrow \varepsilon$

- The rules will be pushed onto the stack – when a variable A is on top of the stack and if there is a rule $A \rightarrow w$, you pop A and push w
- You can go to the accept state only if the stack is empty

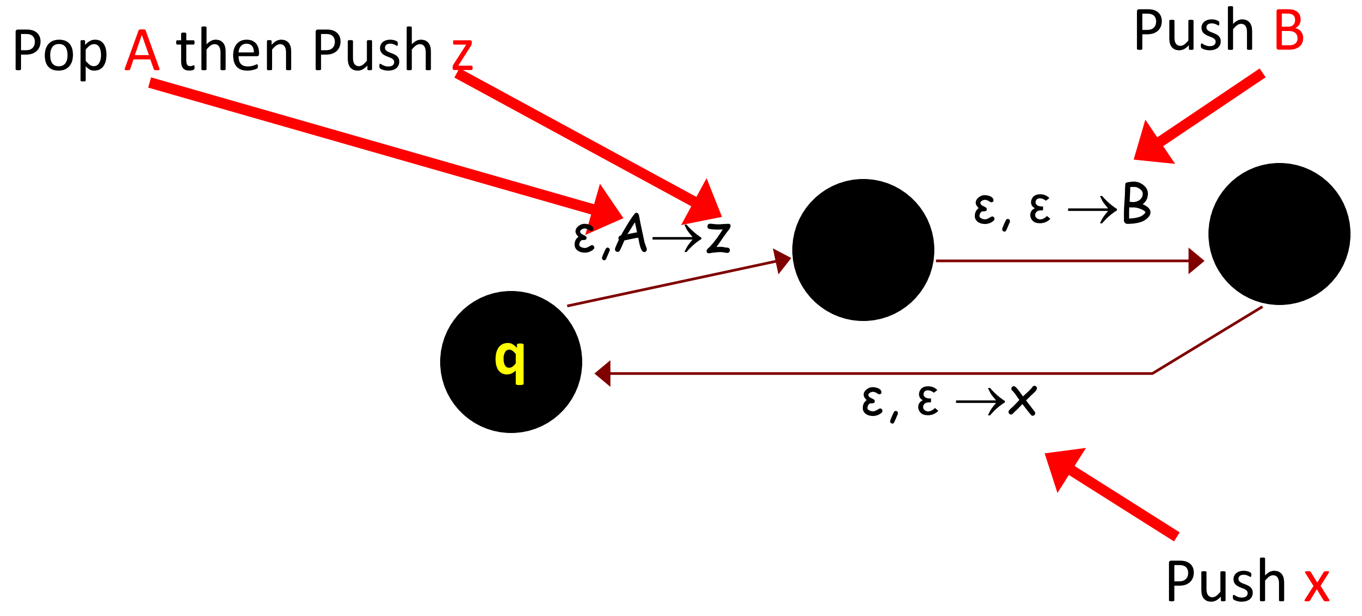
Idea of PDA construction for A→xBz



If the symbol in the LHS is at the top of stack, pop it

and push the symbols in the RHS to stack

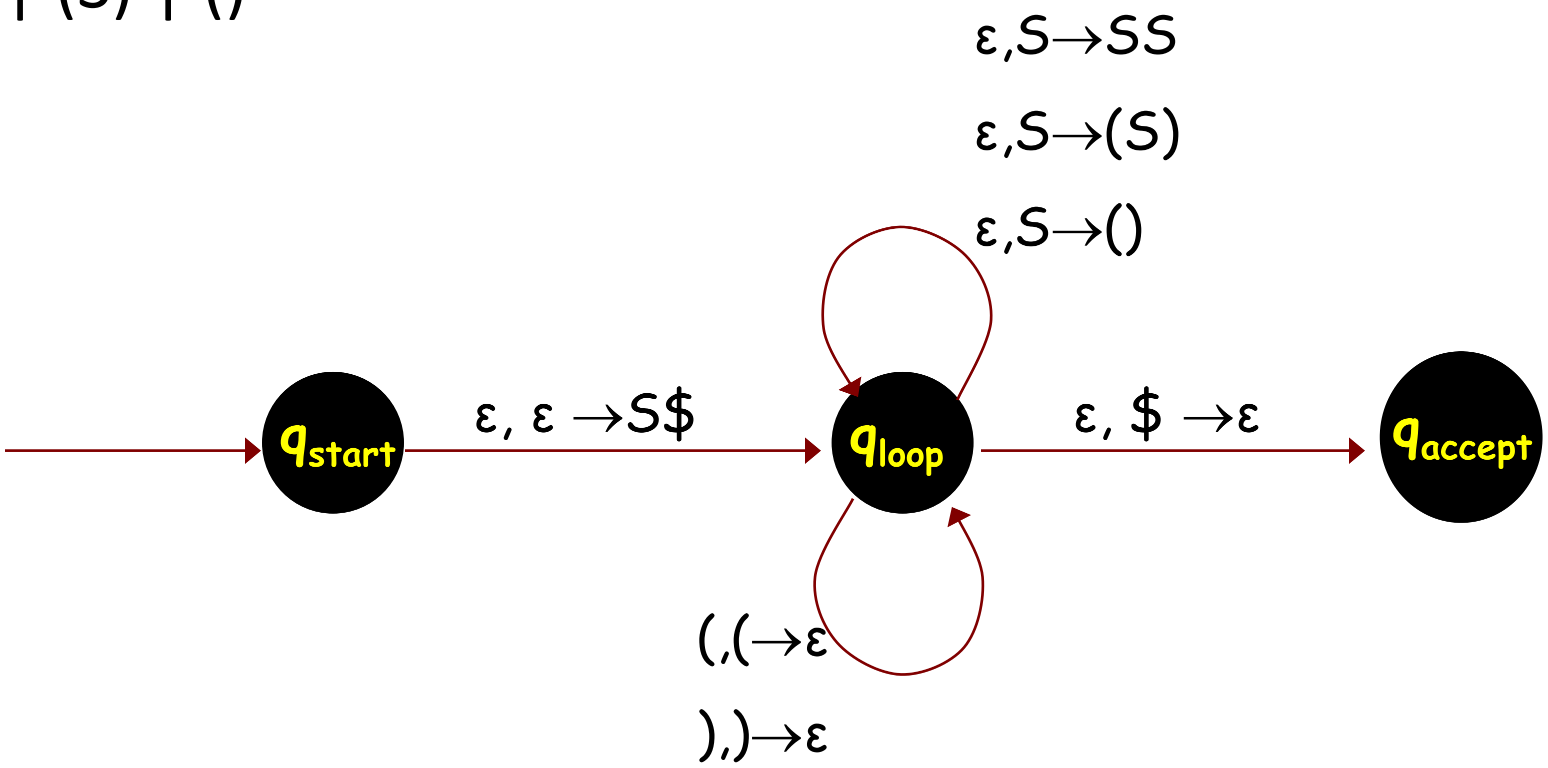
Actual construction for $A \rightarrow xBz$



we say $\delta(q,\varepsilon,A)=(q,xBz)$

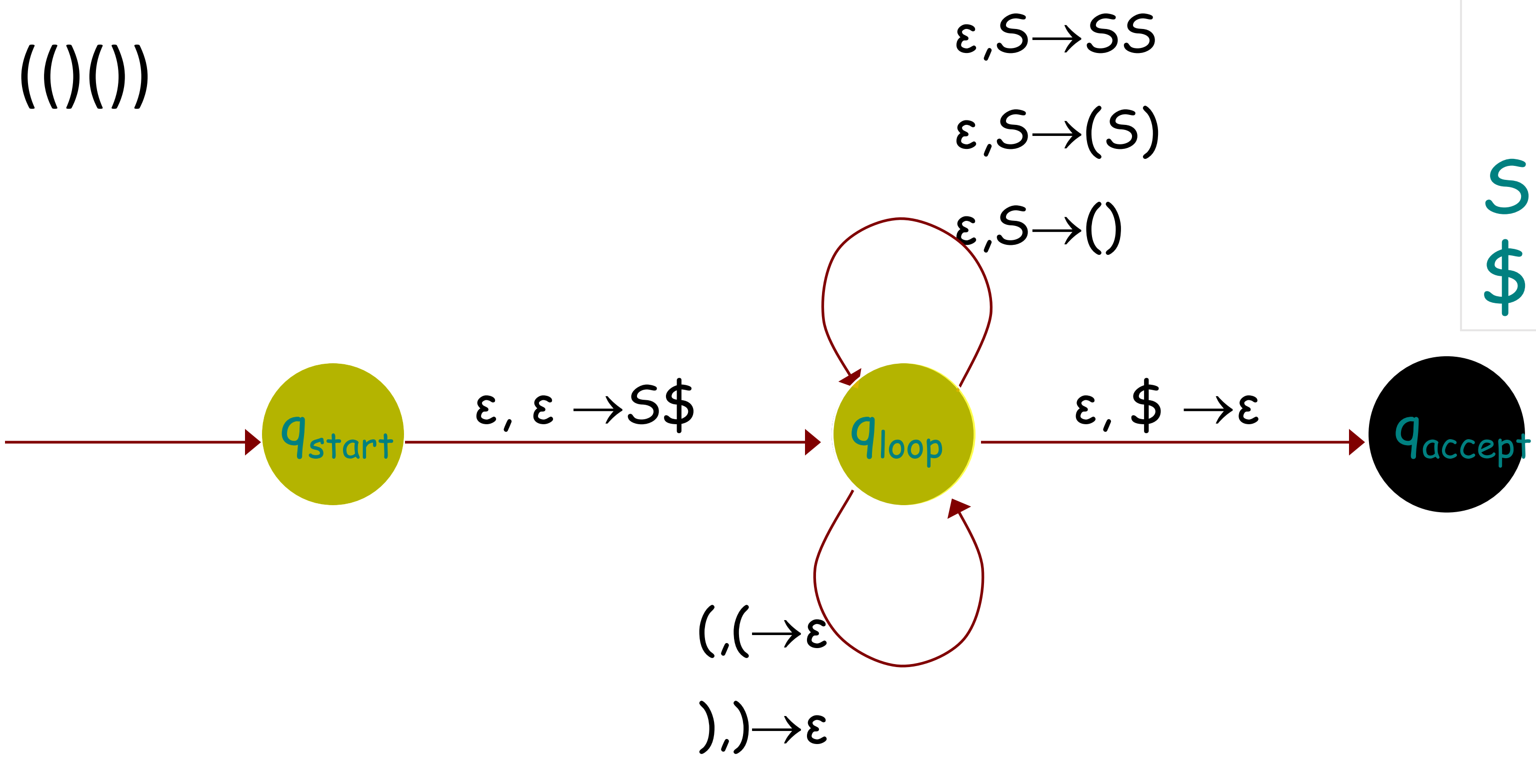
Example

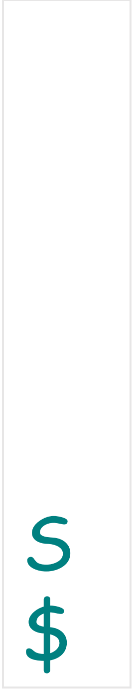
- $S \rightarrow SS \mid (S) \mid ()$



Example

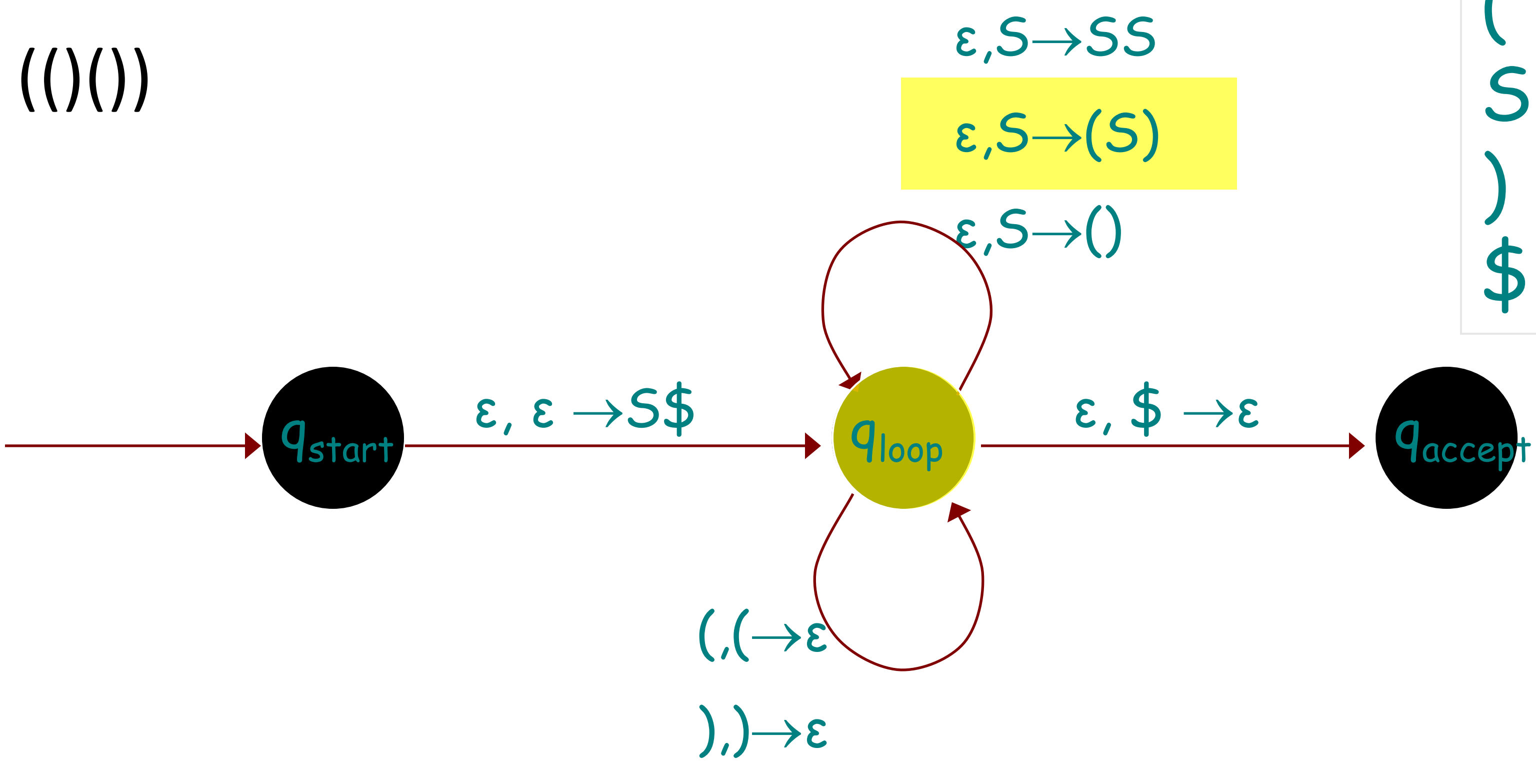
- Read (()(()))

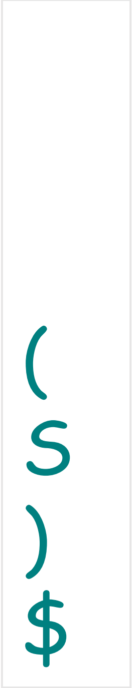




Example

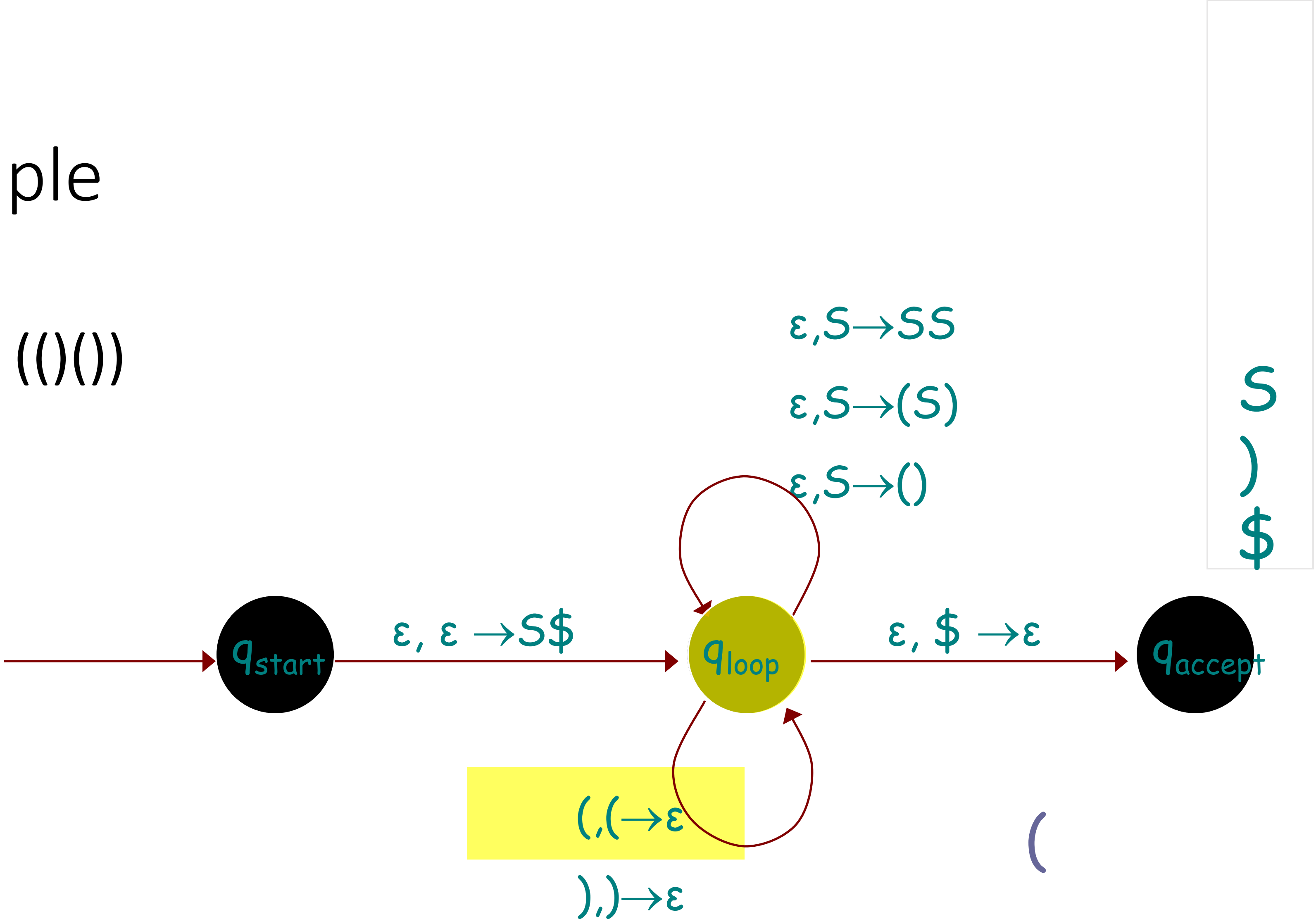
- Read (()(())





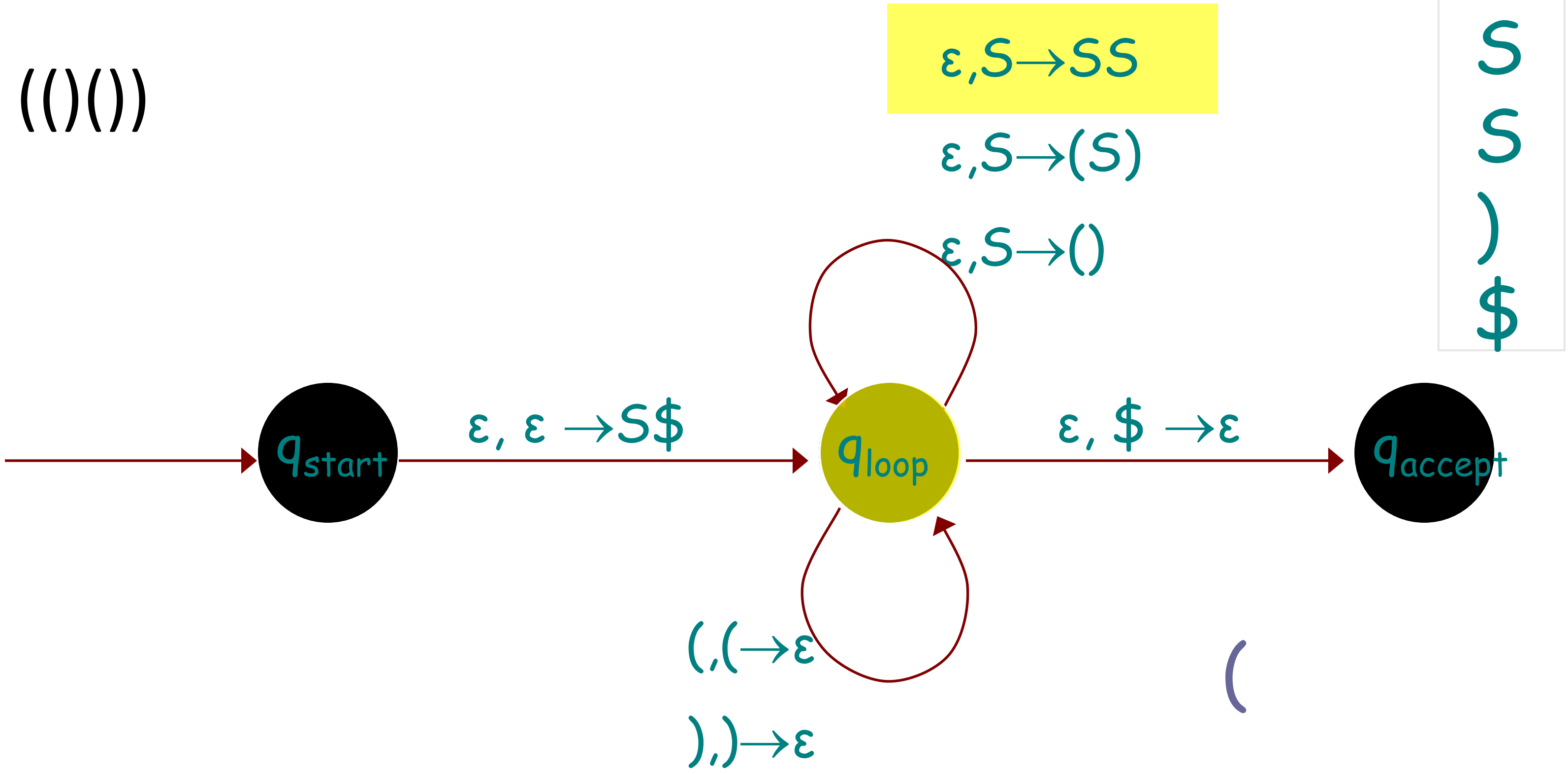
Example

- Read (()(())



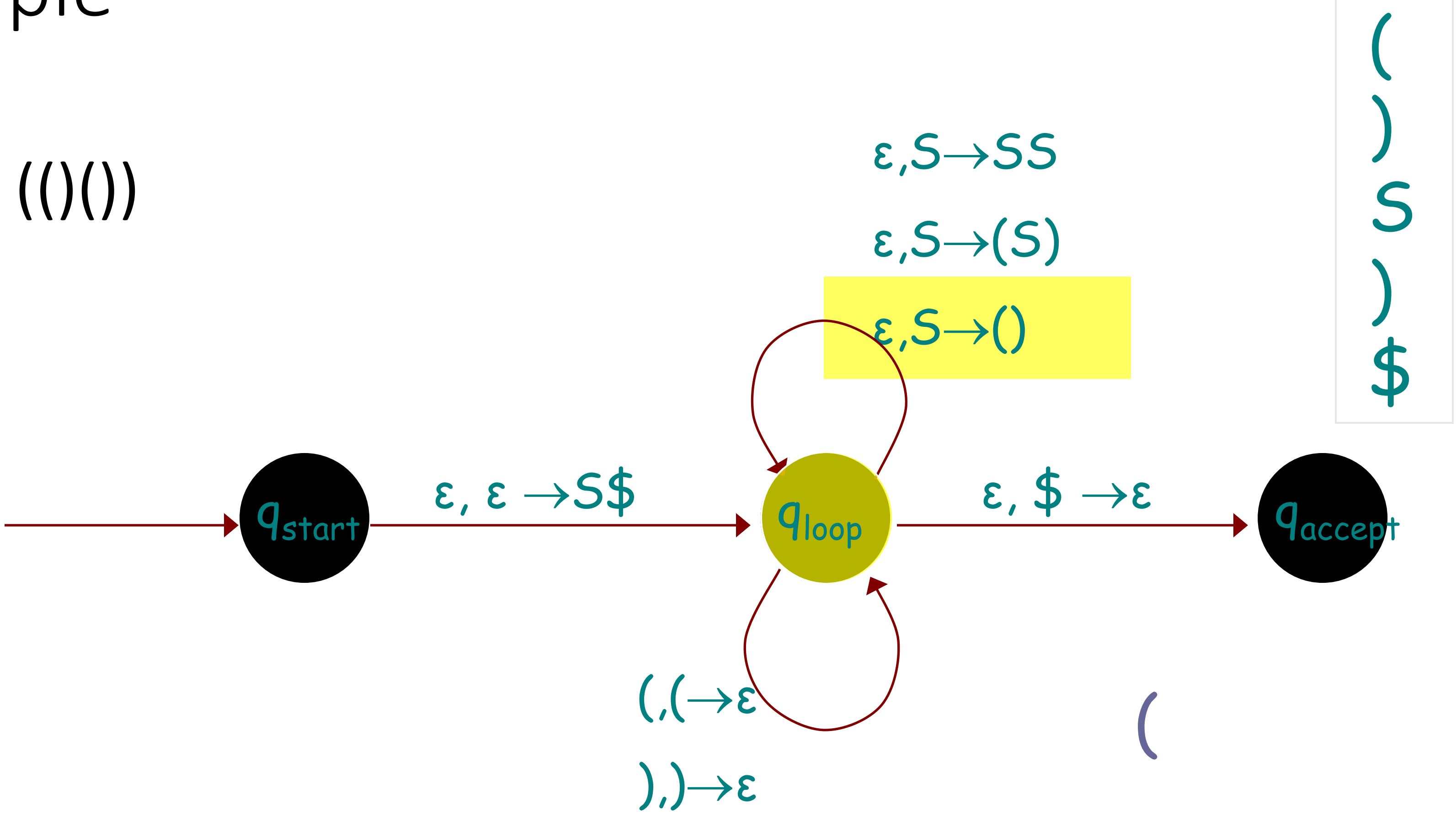
Example

- Read (()())



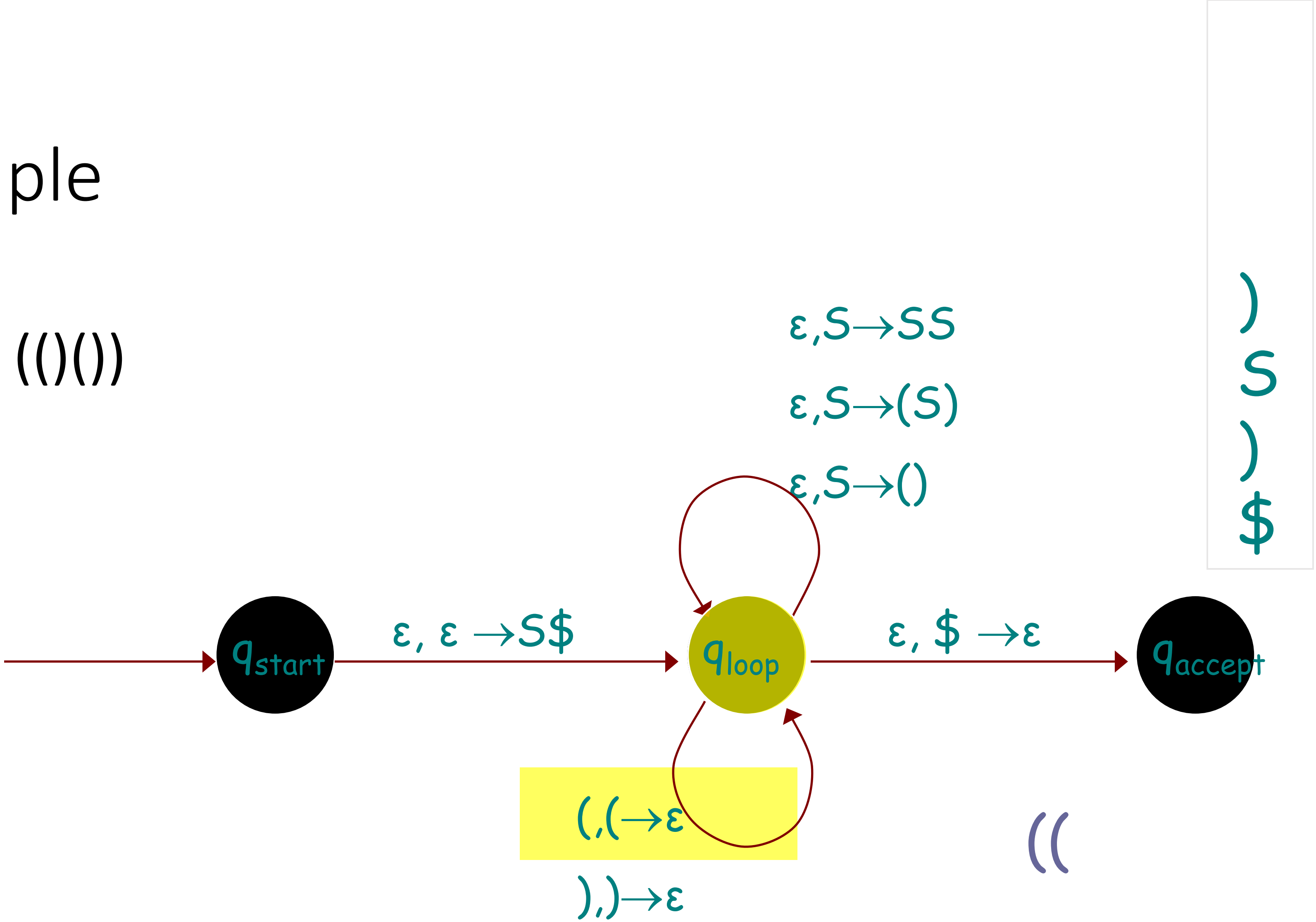
Example

- Read (()())



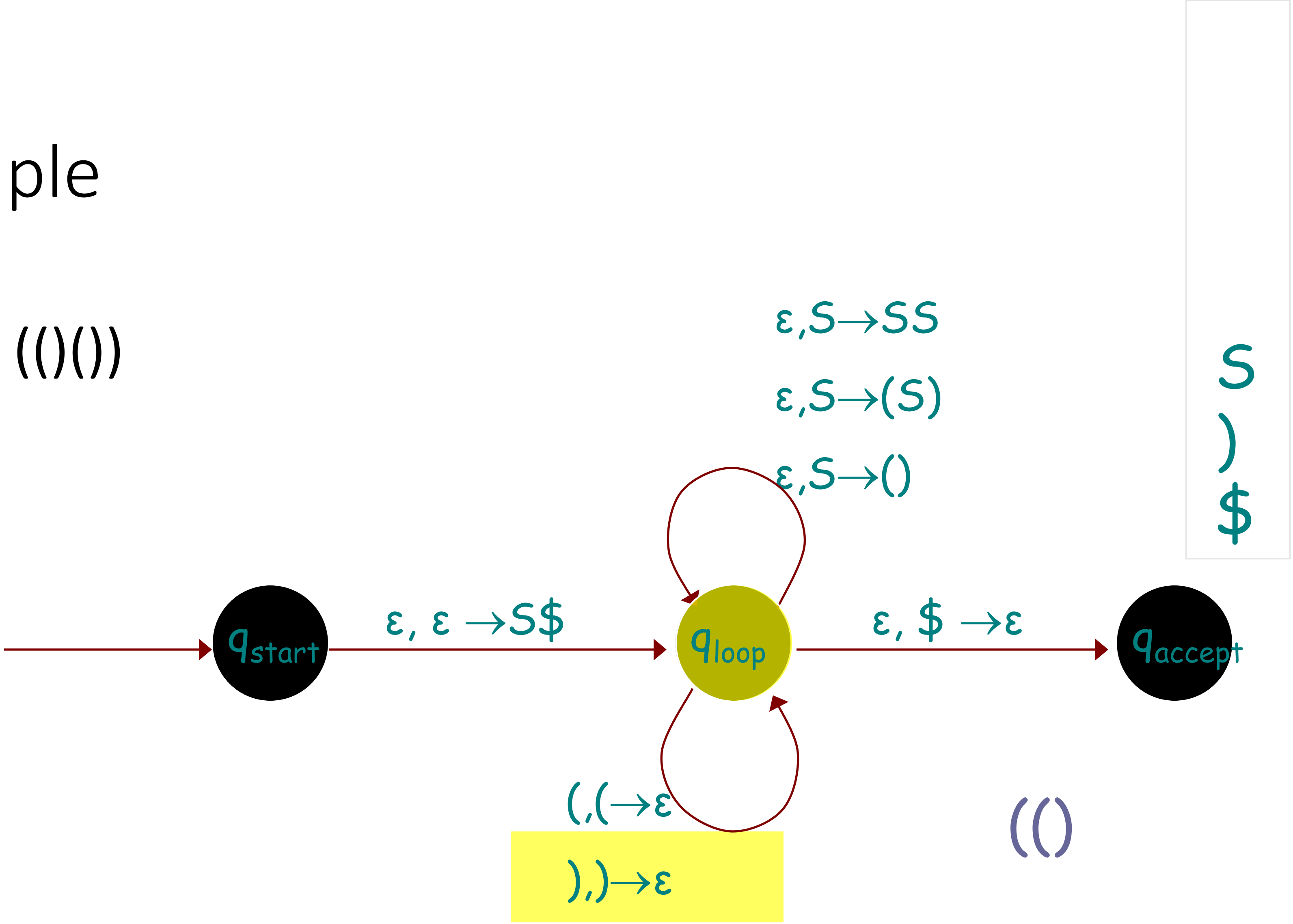
Example

- Read (()())



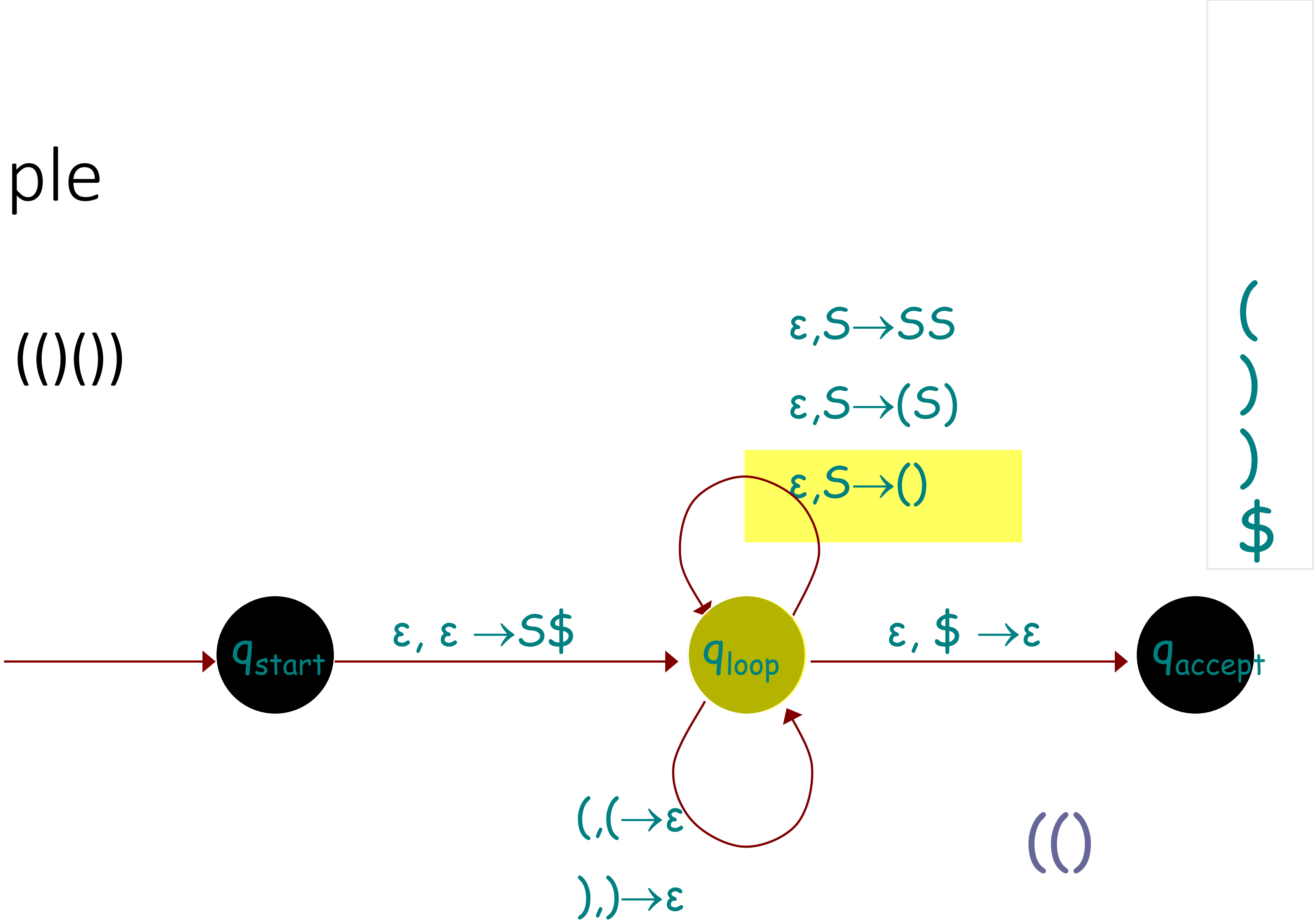
Example

- Read (()(())



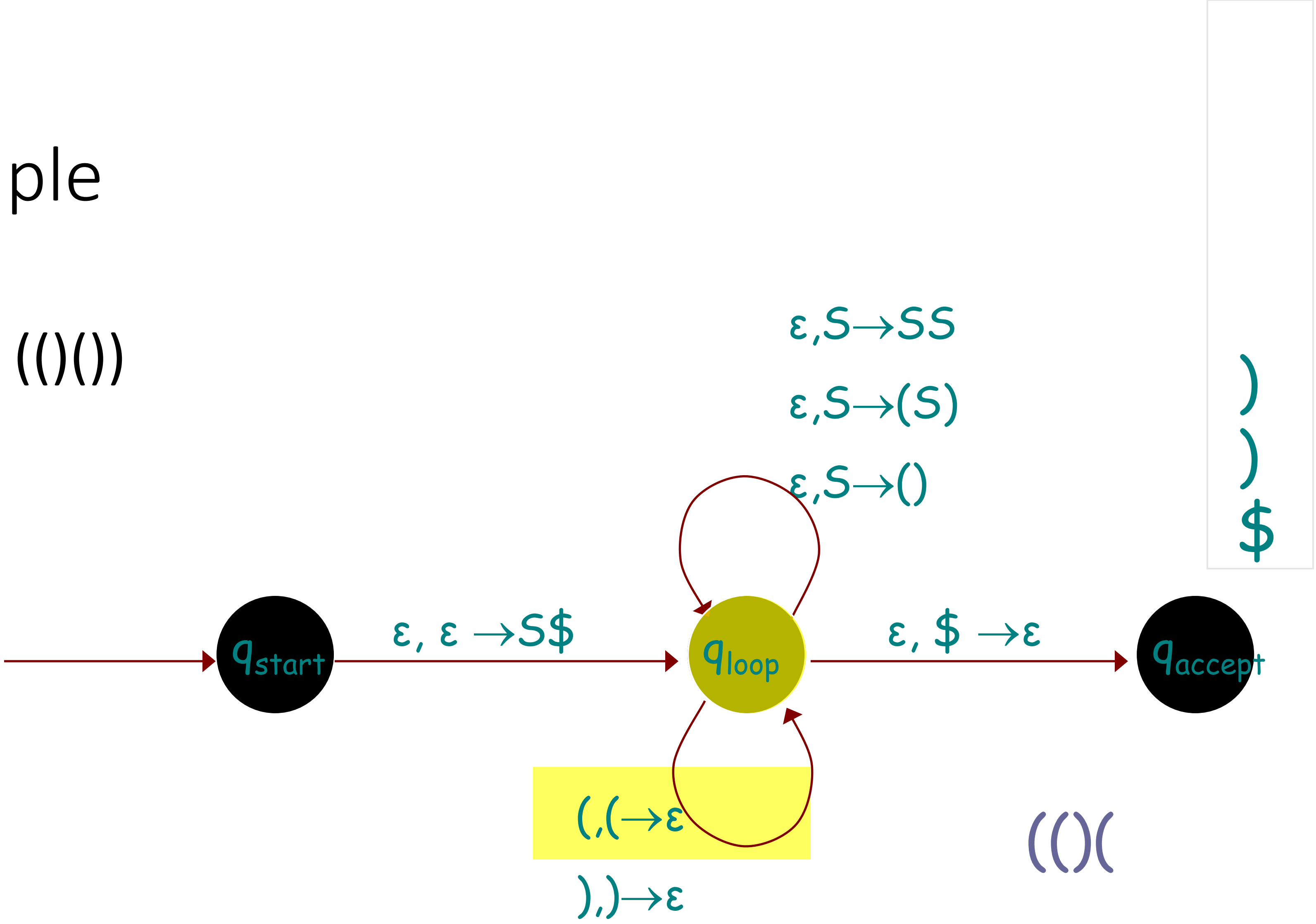
Example

- Read (()(())



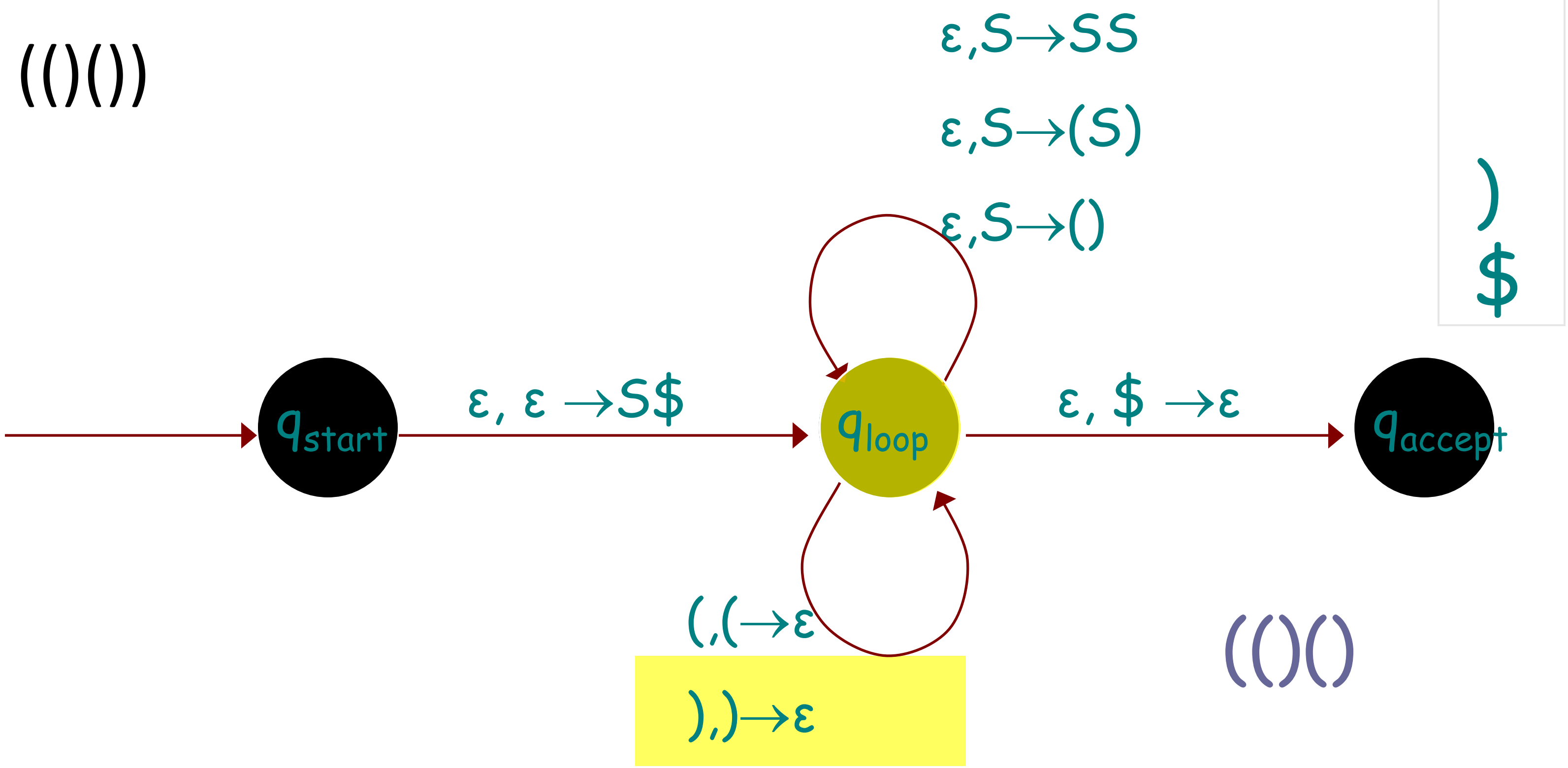
Example

- Read (()())



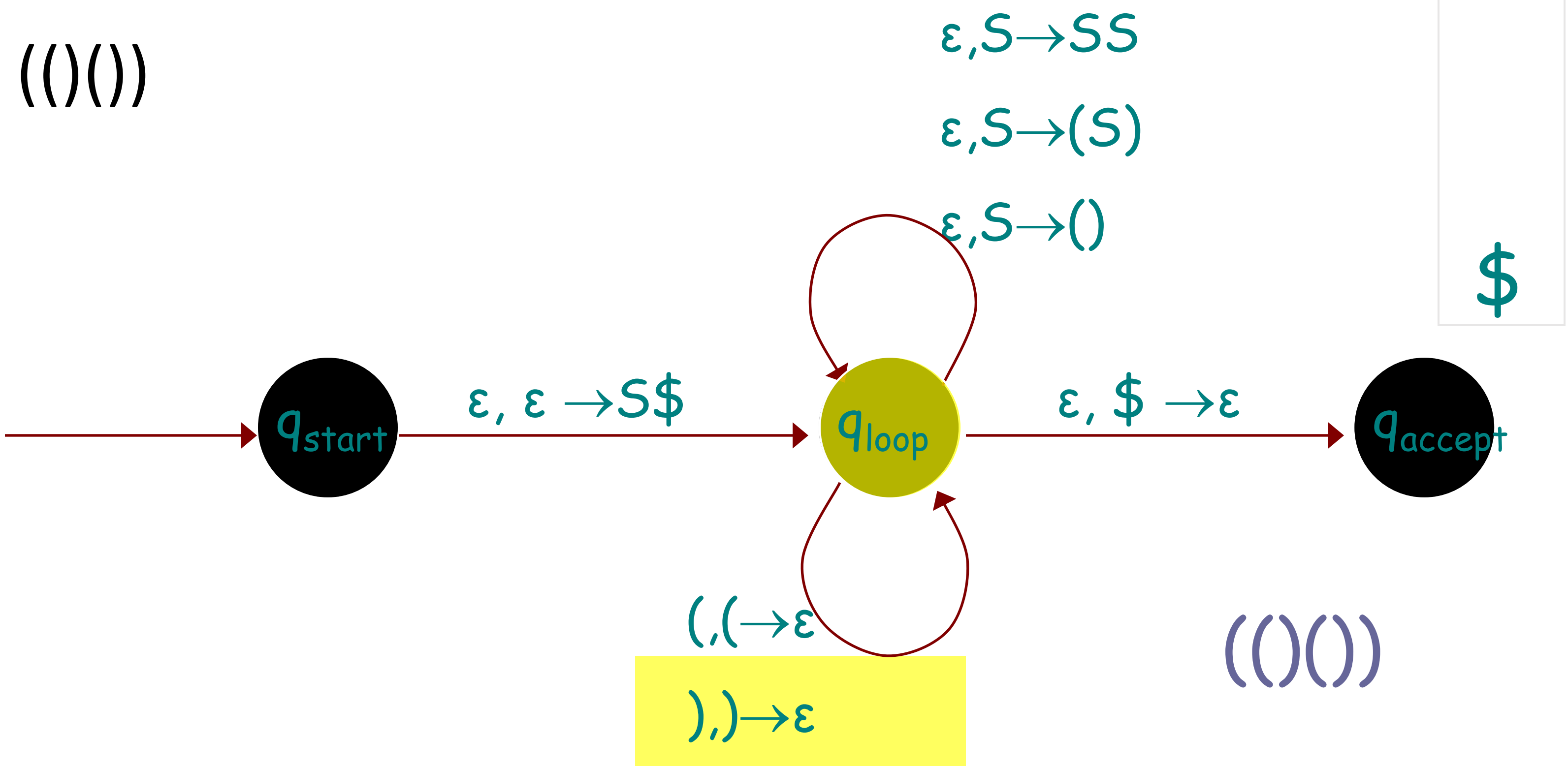
Example

- Read (()(())



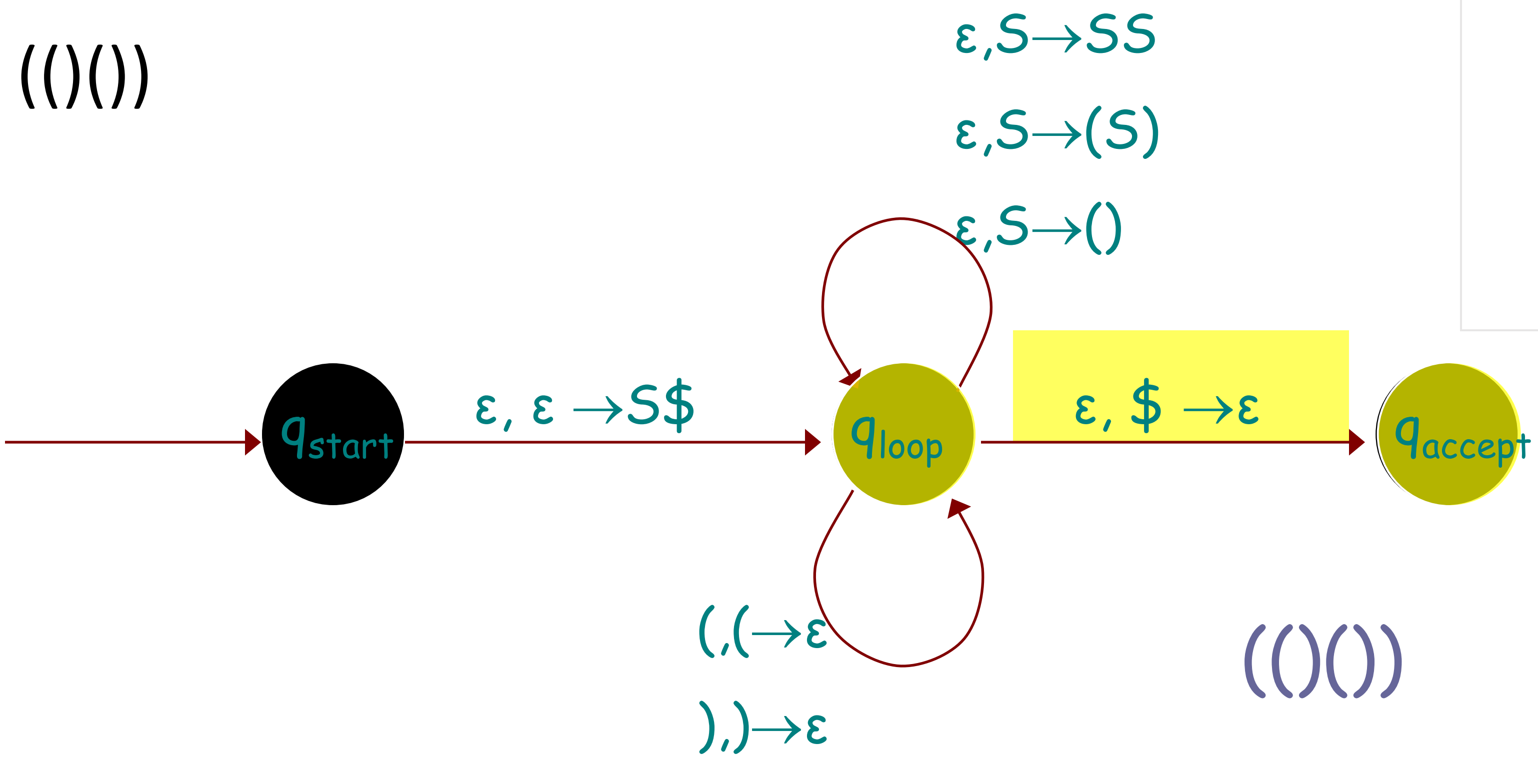
Example

- Read (()())



Example

- Read (()())



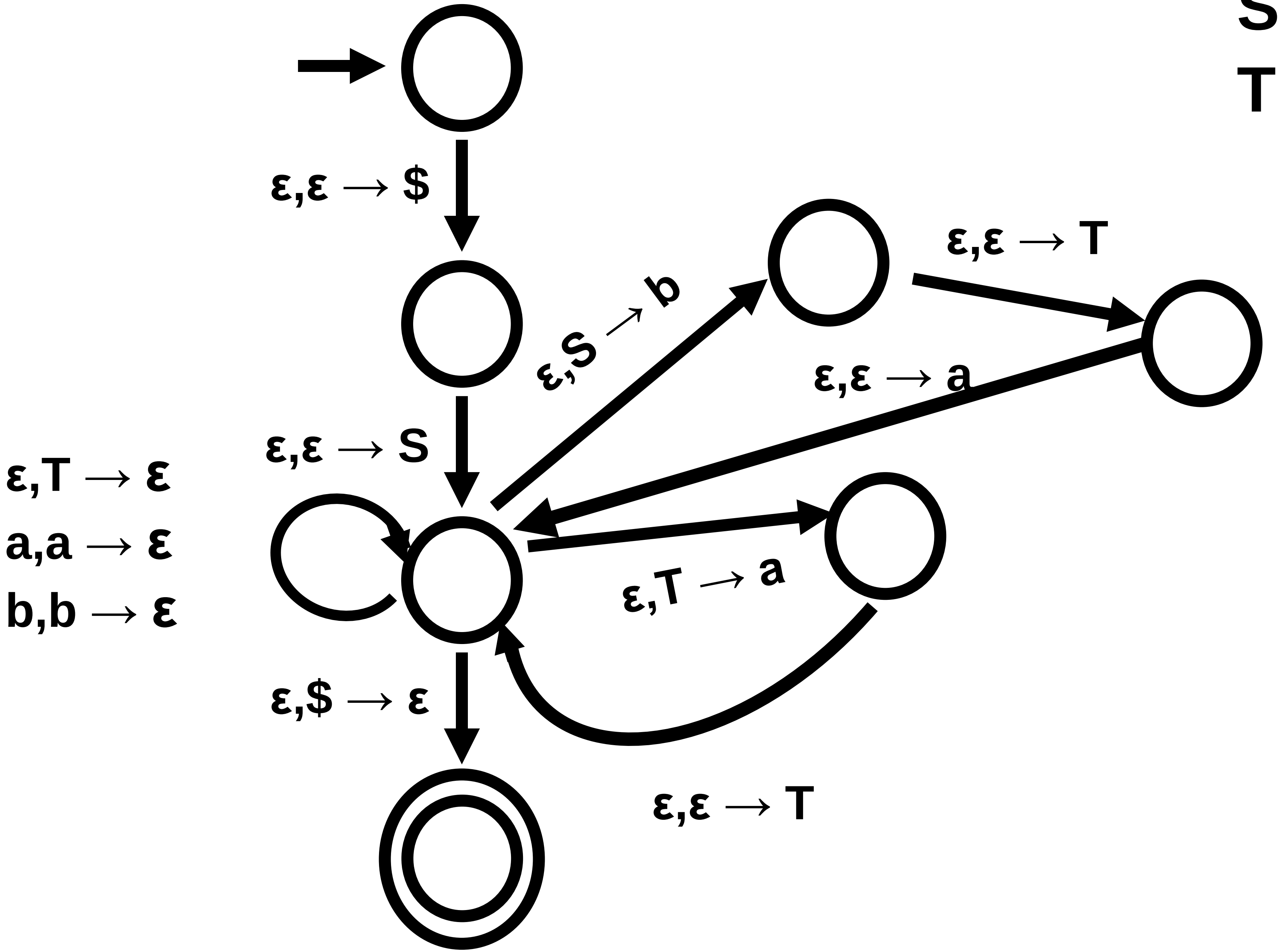
A Language L is generated by a CFG If L is recognized by a PDA

Suppose L is generated by a CFG $G = (V, \Sigma, R, S)$

Construct $P = (Q, \Sigma, \Gamma, \delta, q, F)$ that recognizes L

$S \to aTb$

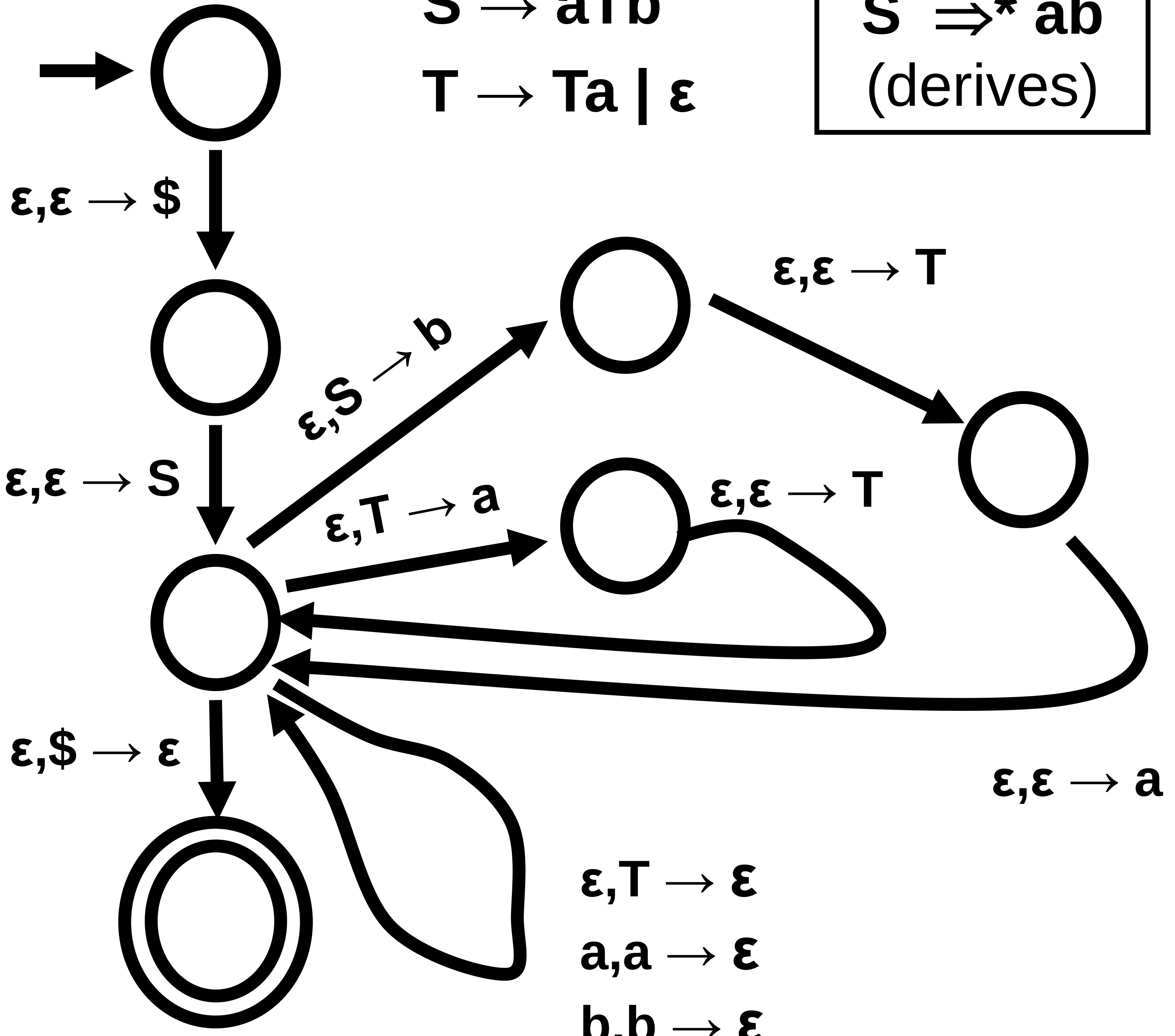
$T \to Ta \mid \varepsilon$



S → aTb

T → Ta | ε

S ⇒* ab
(derives)



Suppose L is generated by a CFG G = (V, Σ, R, S)

Describe P = (Q, Σ, Γ, δ, q, F) that recognizes L :

(1) Push $ and then S on the stack

(2) Repeat the following steps forever:

(a) Pop the stack, call the result X.

(b) If X is a variable A, guess a rule that matches A and push result into the stack

(c) If X is a terminal, read next symbol from input and compare it to terminal. If they’re different, *reject*.

(d) If X is $: then *accept* iff no more input