CS 302.1 - Automata Theory

Lecture 06

Shantanav Chakraborty

Center for Quantum Science and Technology (CQST)

Center for Security, Theory and Algorithms (CSTAR)

IIIT Hyderabad

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

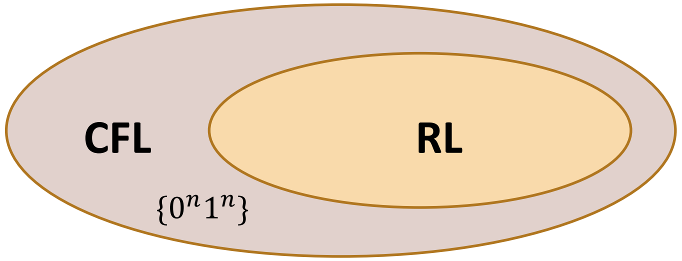
HYDERABAD

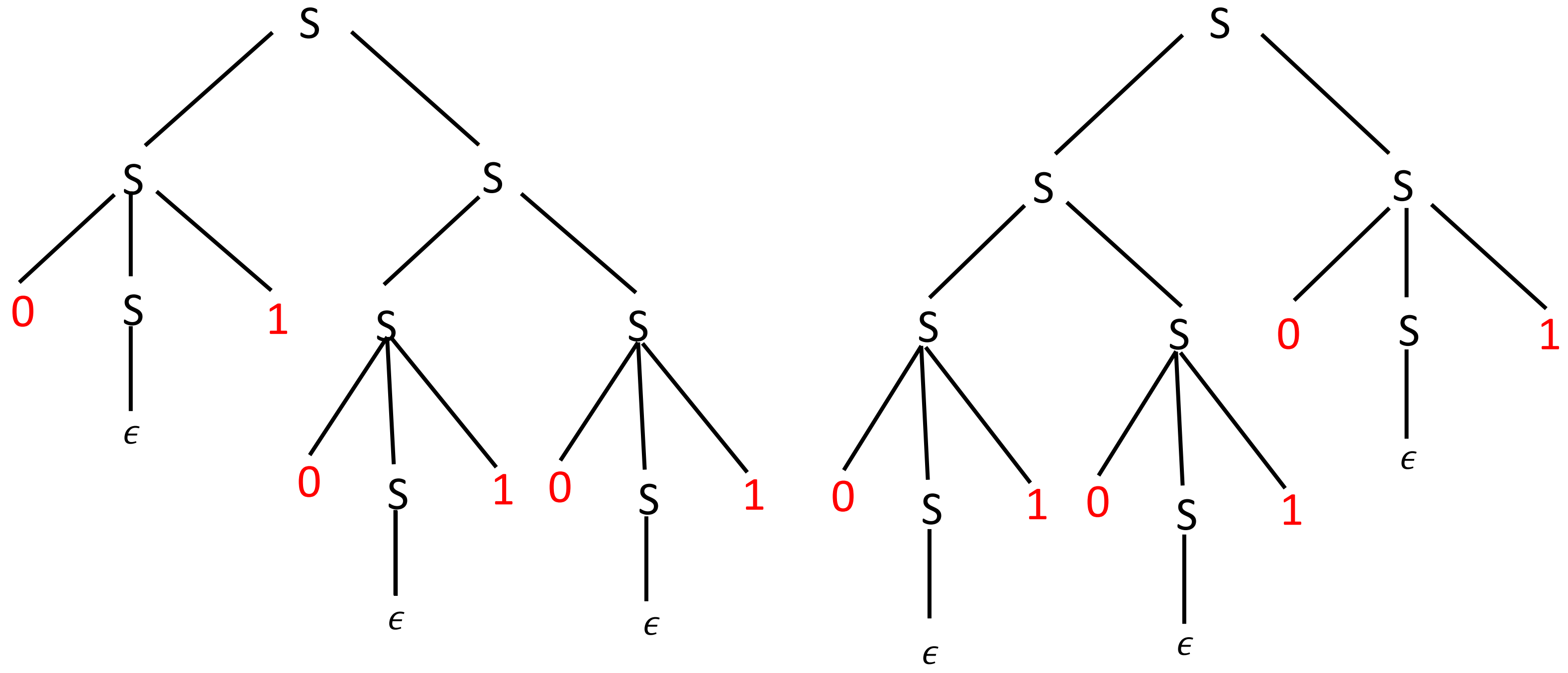
Quick Recap

Context-Free Grammars: If the *rules* of the underlying grammar G are of the form

$$\boldsymbol{V} \to (\boldsymbol{V} \cup \boldsymbol{T})^*$$

then such a grammar is called **Context-Free.**





Parse trees: These are ordered trees that provide alternative representations of the derivation of a grammar.

Ambiguous grammars: There exists $\boldsymbol{\omega} \in \boldsymbol{L}(\boldsymbol{G})$, such that there are **two or more leftmost derivations for $\boldsymbol{\omega}$** (or equivalently two or more rightmost derivations) or equivalently **two or more parse trees for $\boldsymbol{\omega}$. Ambiguity** may not be desirable

Quick Recap

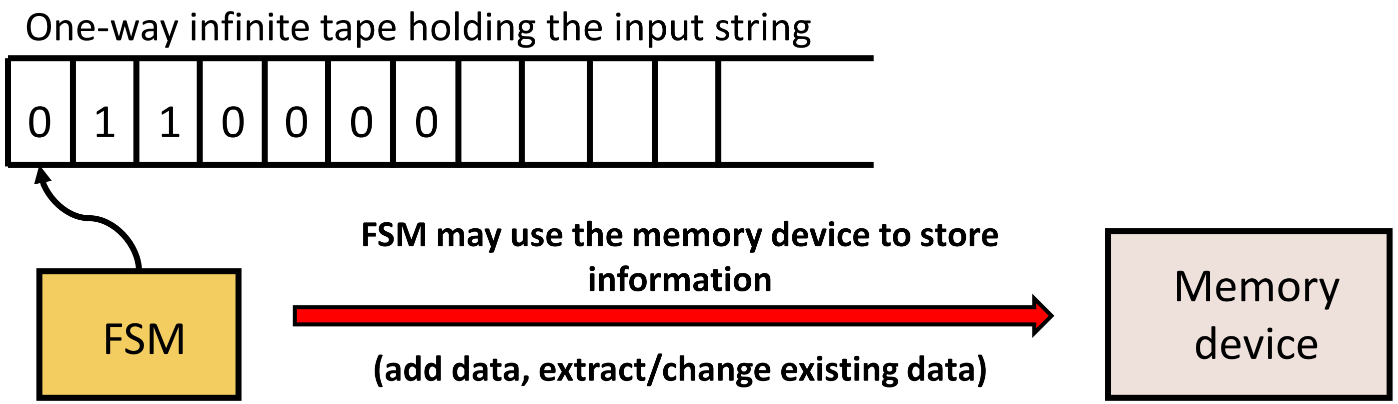
Chomsky Normal Form: If every *rule* of the CFG is of the form

$$\boldsymbol{A \to BC} \qquad [B, C \text{ are not start variables}]$$
$$\boldsymbol{A \to a} \qquad [a \text{ is a terminal}]$$
$$\boldsymbol{S \to \epsilon} \qquad [S \text{ is the Start Variable}]$$

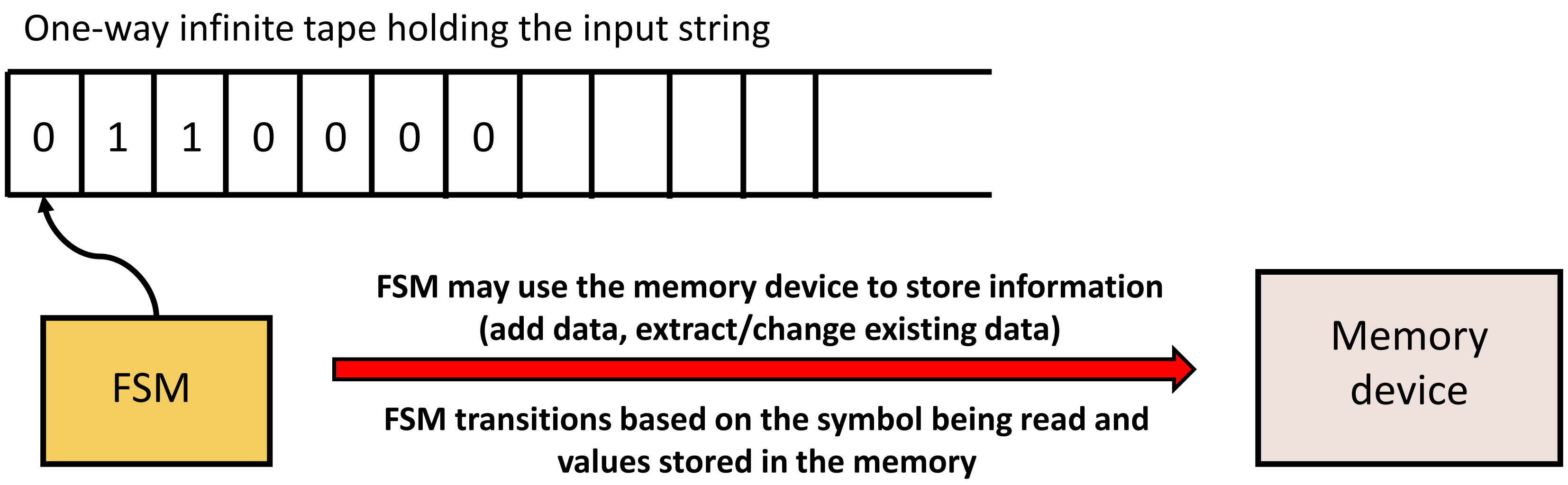
- Any CFG can be converted to a grammar in CNF that generates the same language.
- The number of steps required to derive a string $w = 2|w| - 1$.
- Is crucial in deciding whether w is generated by a CFG G.

Pushdown Automata

- Automata that recognizes CFLs
- FSM + memory device
- FSM transitions by reading an input symbol and by interacting with the device



Pushdown Automata



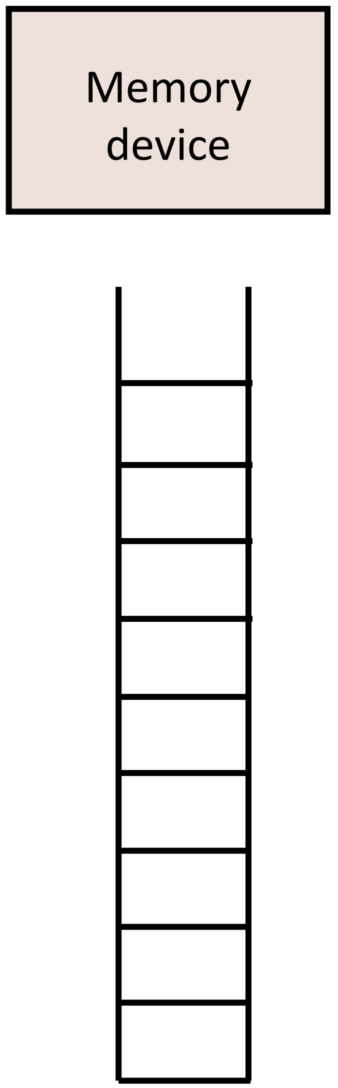
The memory device

- Simple memory device with unbounded memory.

Pushdown Automata

The memory device

- Simple memory device with unbounded memory.
- Consider a **STACK**
- At any stage, new elements can be added to the Stack (**PUSH**).
- At any stage, the element at the **top** of the STACK can be read by removing it from the stack (**POP**).



Pushdown Automata

The memory device

- Simple memory device with unbounded memory.
- Consider a **STACK**
- At any stage, elements can be pushed or popped.

PUSH

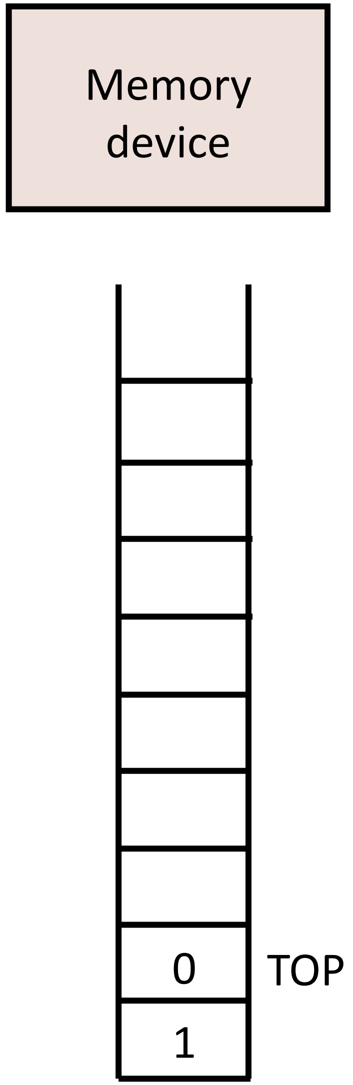
- New symbols can be **pushed** in to the STACK.

  E.g: PUSH 1

- The Top of the STACK now covers the old stack top, i.e.

  TOP = TOP + 1

- The size of the stack keeps growing.



Pushdown Automata

The memory device

- Simple memory device with unbounded memory.
- Consider a **STACK**
- At any stage, elements can be pushed or popped.

PUSH

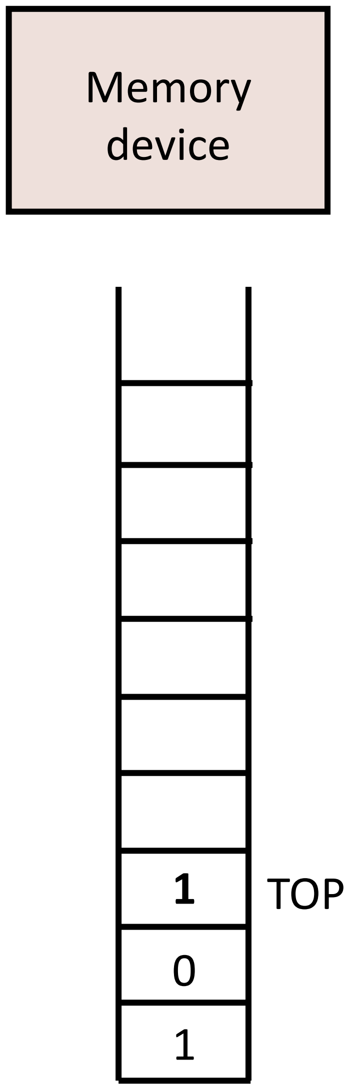
- New symbols can be **pushed** in to the STACK.

  E.g: PUSH 1

- The Top of the STACK now covers the old stack top, i.e.

  TOP = TOP + 1

- The size of the stack keeps growing.



Pushdown Automata

The memory device

- Simple memory device with unbounded memory.
- Consider a **STACK**
- At any stage, elements can be pushed or popped.

PUSH

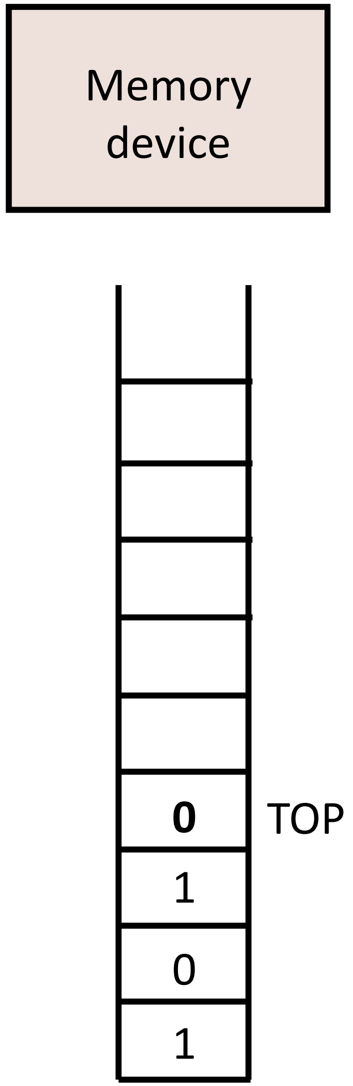
- New symbols can be **pushed** in to the STACK.

  E.g: **PUSH 0**

- The Top of the STACK now covers the old stack top, i.e.

  TOP = TOP + 1

- The size of the stack keeps growing.



Pushdown Automata

The memory device

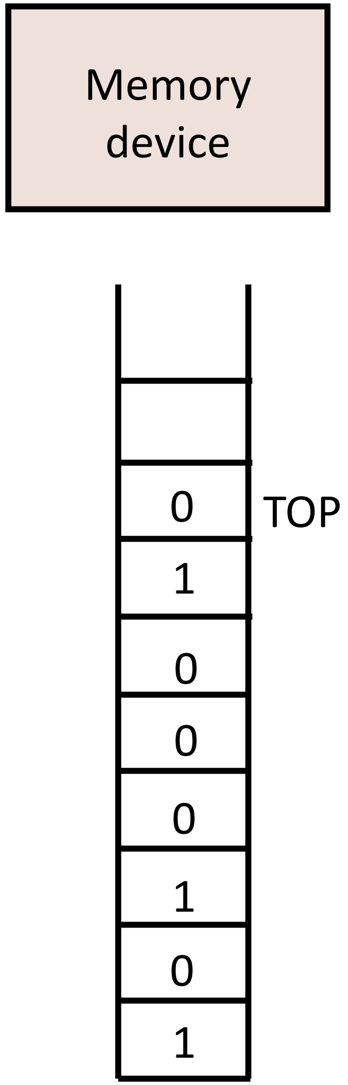
- Simple memory device with unbounded memory.
- Consider a **STACK**
- At any stage, elements can be pushed or popped.

PUSH

- New symbols can be **pushed** in to the STACK.
- The Top of the STACK now covers the old stack top, i.e.

  TOP = TOP + 1

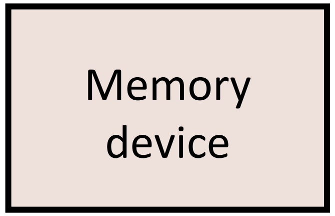
- The size of the stack keeps growing.



Pushdown Automata

The memory device

- Simple memory device with unbounded memory.
- Consider a **STACK**
- At any stage, elements can be pushed or popped.



POP

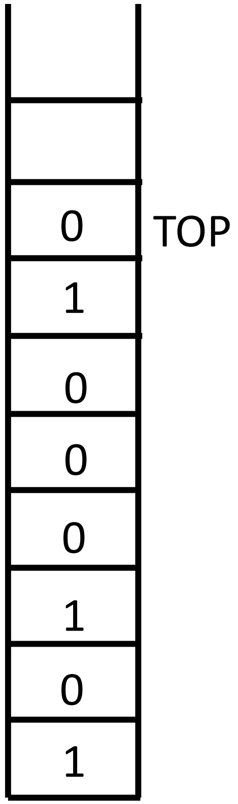
- The element from the TOP of the stack can be **popped** out

E.g.: **POP 0**

- The Top of the STACK moves to the element below.

TOP = TOP - 1

- Successive **POP** operations shrink the stack size. Elements can be popped until EMPTY.
- **Last In First Out (LIFO)**: The last element that was pushed is the first to be popped out



Pushdown Automata

The memory device

- Simple memory device with unbounded memory.
- Consider a **STACK**
- At any stage, elements can be pushed or popped.

Memory device

POP

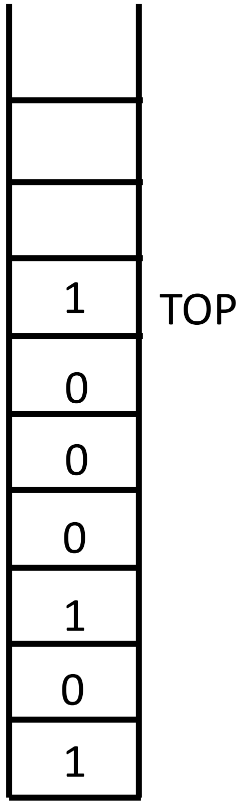
- The element from the TOP of the stack can be **popped** out

E.g.: **POP 0**

- The Top of the STACK moves to the element below.

TOP = TOP - 1

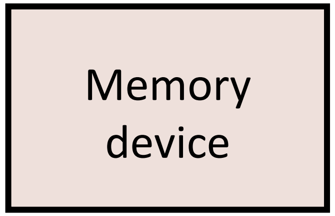
- Successive **POP** operations shrink the stack size. Elements can be popped until EMPTY.
- **Last In First Out (LIFO)**: The last element that was pushed is the first to be popped out



Pushdown Automata

The memory device

- Simple memory device with unbounded memory.
- Consider a **STACK**
- At any stage, elements can be pushed or popped.



POP

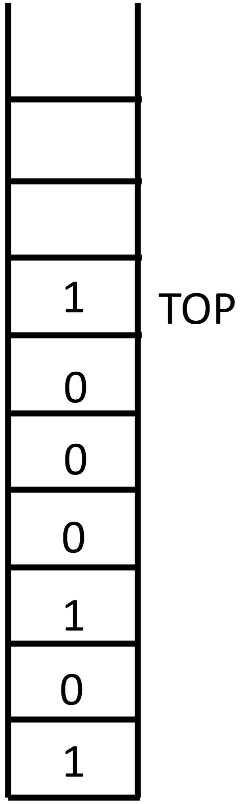
- The element from the TOP of the stack can be **popped** out

E.g.: **POP 1**

- The Top of the STACK moves to the element below.

TOP = TOP - 1

- Successive **POP** operations shrink the stack size. Elements can be popped until EMPTY.
- **Last In First Out (LIFO)**: The last element that was pushed is the first to be popped out



Pushdown Automata

The memory device

- Simple memory device with unbounded memory.
- Consider a **STACK**
- At any stage, elements can be pushed or popped.

POP

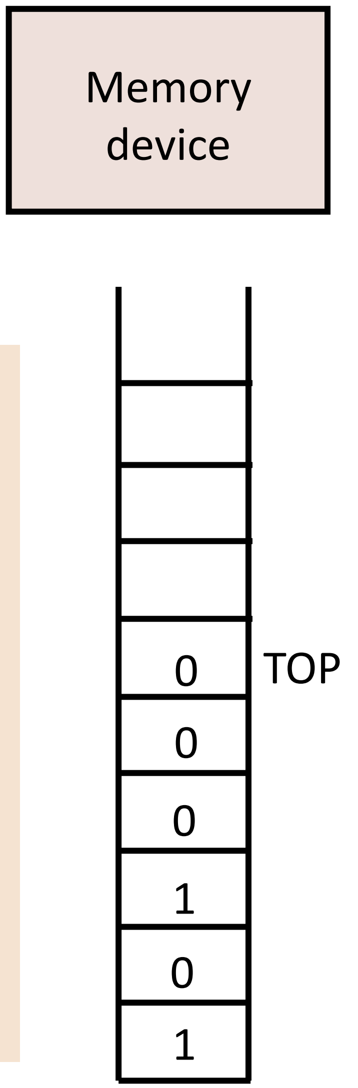
- The element from the TOP of the stack can be **popped** out

E.g.: **POP 1**

- The Top of the STACK moves to the element below.

TOP = TOP - 1

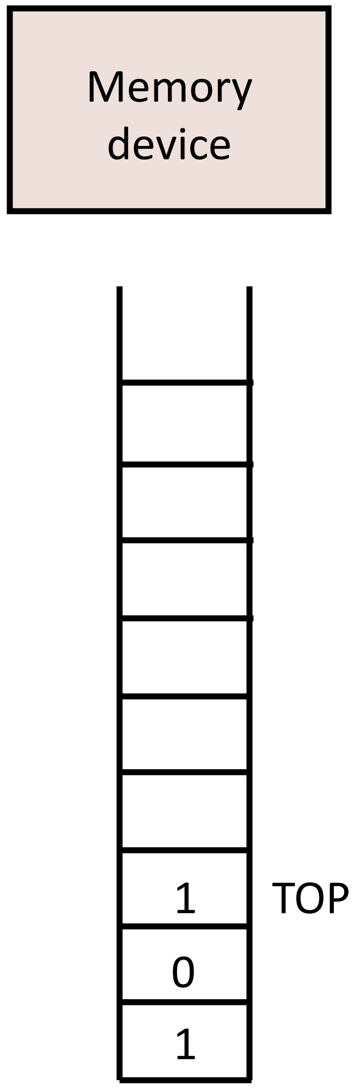
- Successive **POP** operations shrink the stack size. Elements can be popped until EMPTY.
- **Last In First Out (LIFO)**: The last element that was pushed is the first to be popped out



Pushdown Automata

The memory device

- Simple memory device with unbounded memory.
- Consider a **STACK**
- Last In First Out (**LIFO**)



POP

- The element from the TOP of the stack can be **popped** out.
- TOP = TOP - 1
- Elements can be popped until STACK is EMPTY.

- How would you know that the STACK is EMPTY?

Pushdown Automata

The memory device

- Simple memory device with unbounded memory.
- Consider a **STACK**
- Last In First Out (**LIFO**)

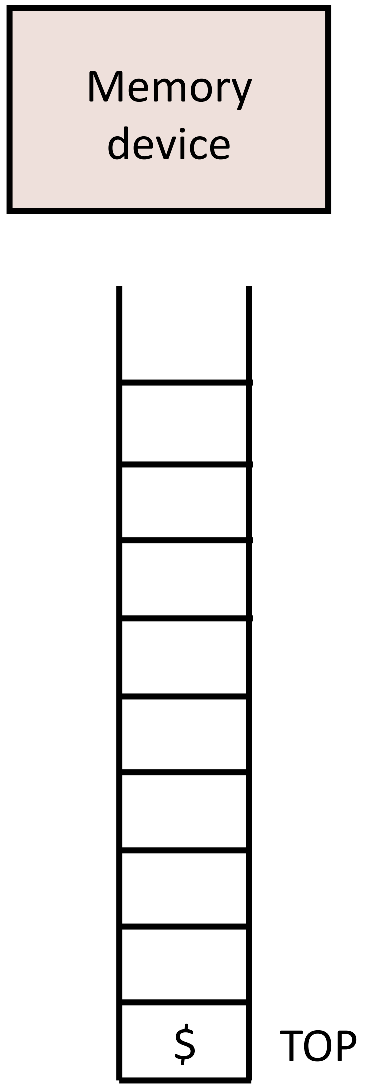
POP

- The element from the TOP of the stack can be **popped** out.
- TOP = TOP - 1
- Elements can be popped until STACK is EMPTY.

- How would you know that the STACK is EMPTY?

- There is generally some special symbol (say $) that demarcates the bottom of the STACK.

- This element is Pushed at the very beginning. Whenever TOP = $, the STACK is EMPTY.



Pushdown Automata

Memory device of PDA: STACK

- STACK is a **LIFO** data structure of unbounded memory
- Only the TOP element can be read from the STACK.
- The bottom of the STACK contains a special symbol ($)
- Characterized by two operations:

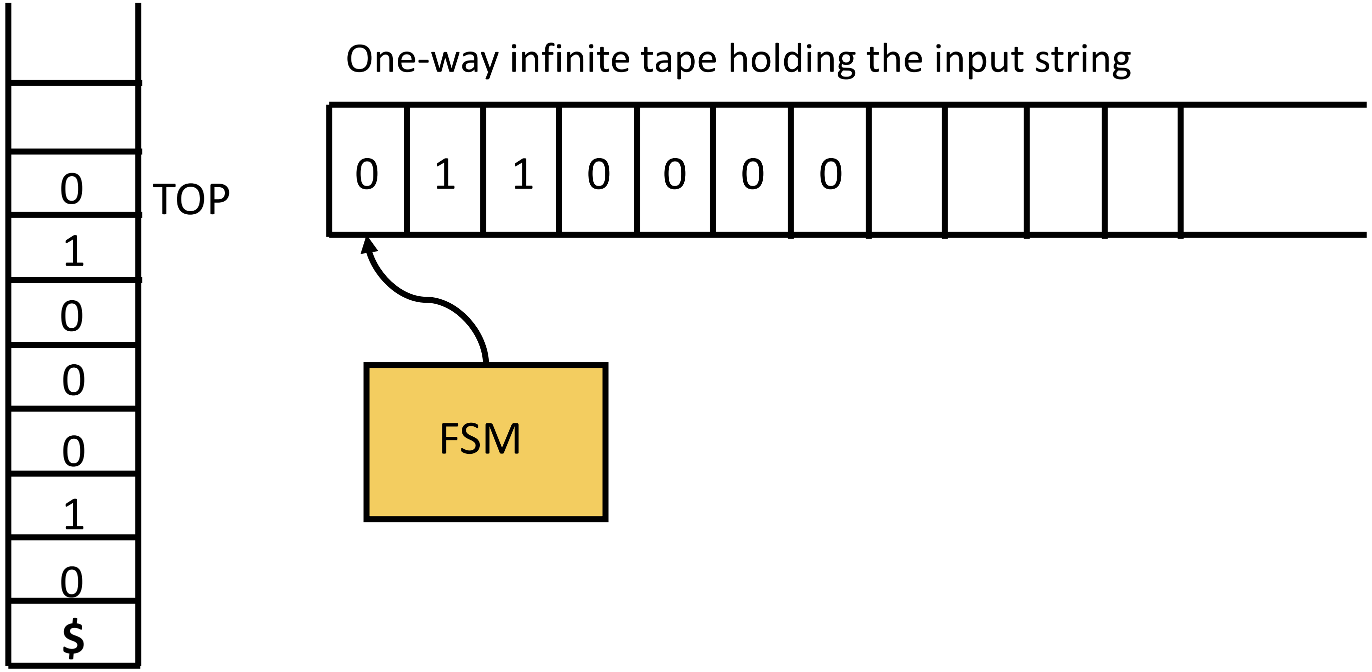
PUSH

- New symbols can be **pushed** in to the STACK.
- TOP = TOP + 1

POP

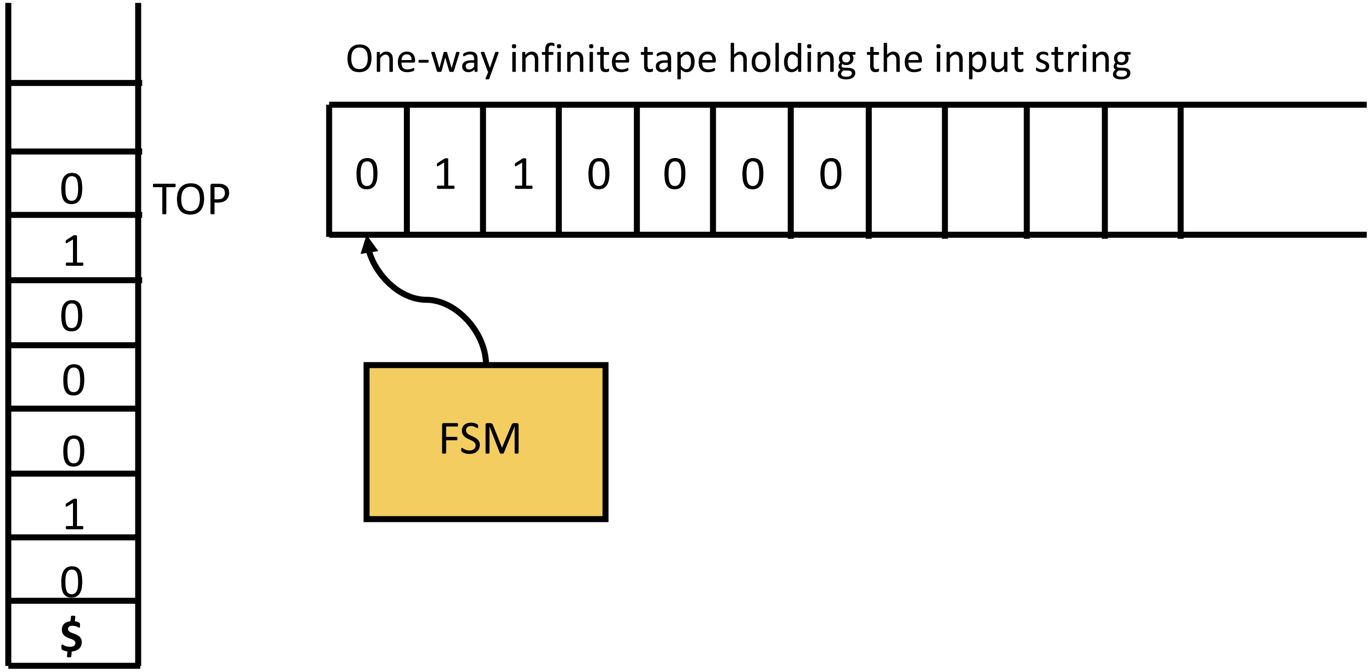
- The element from the TOP of the stack can be **popped** out.
- TOP = TOP - 1
- Elements can be popped until STACK is EMPTY.

Pushdown Automata



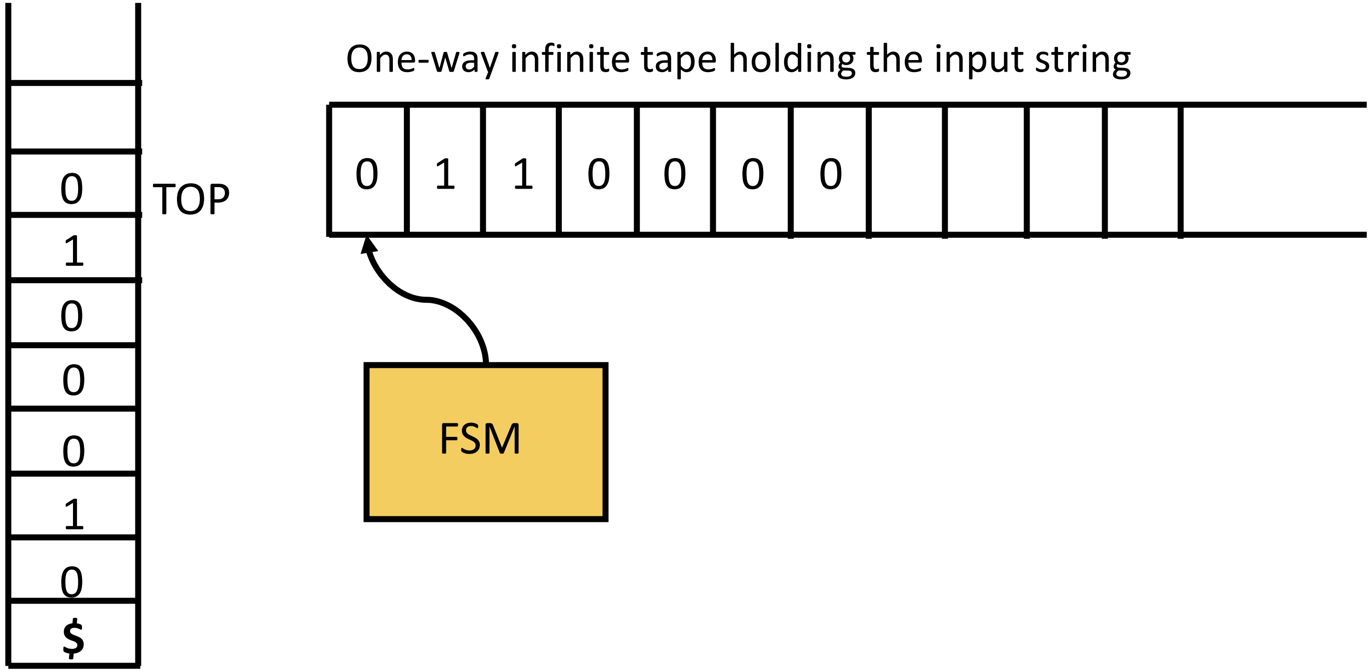
- A Pushdown Automata (PDA) is a finite automaton that has access to a stack.
- The FSM:

Pushdown Automata



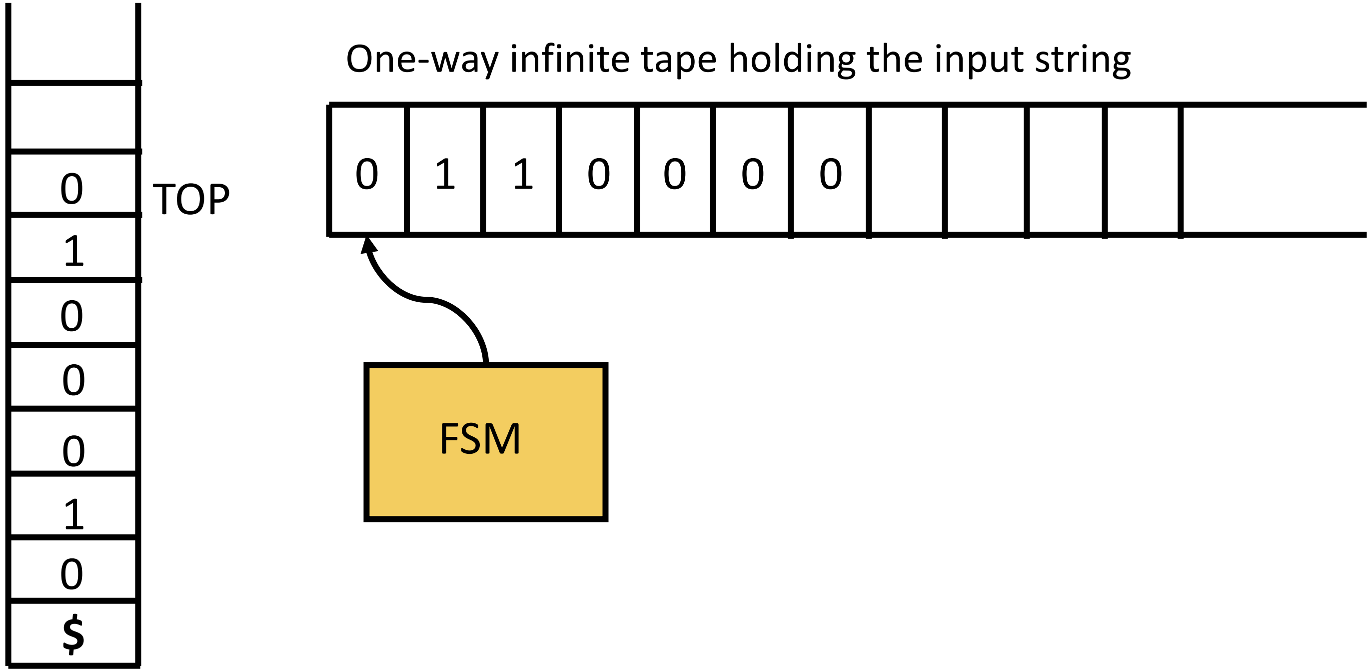
- A Pushdown Automata (PDA) is a finite automaton that has access to a stack.
- The FSM:
  - Transitions based on the Input symbol and the element at the top of the stack (e.g.: If I/P symbol = 0 & TOP = 0, transition from i to j)

Pushdown Automata



- A Pushdown Automata (PDA) is a finite automaton that has access to a stack.
- The FSM:
  - Transitions based on the Input symbol and the element at the top of the stack (e.g.: If I/P symbol = 0 & TOP = 0, transition from i to j).
  - **How can we read the TOP? By popping**

Pushdown Automata

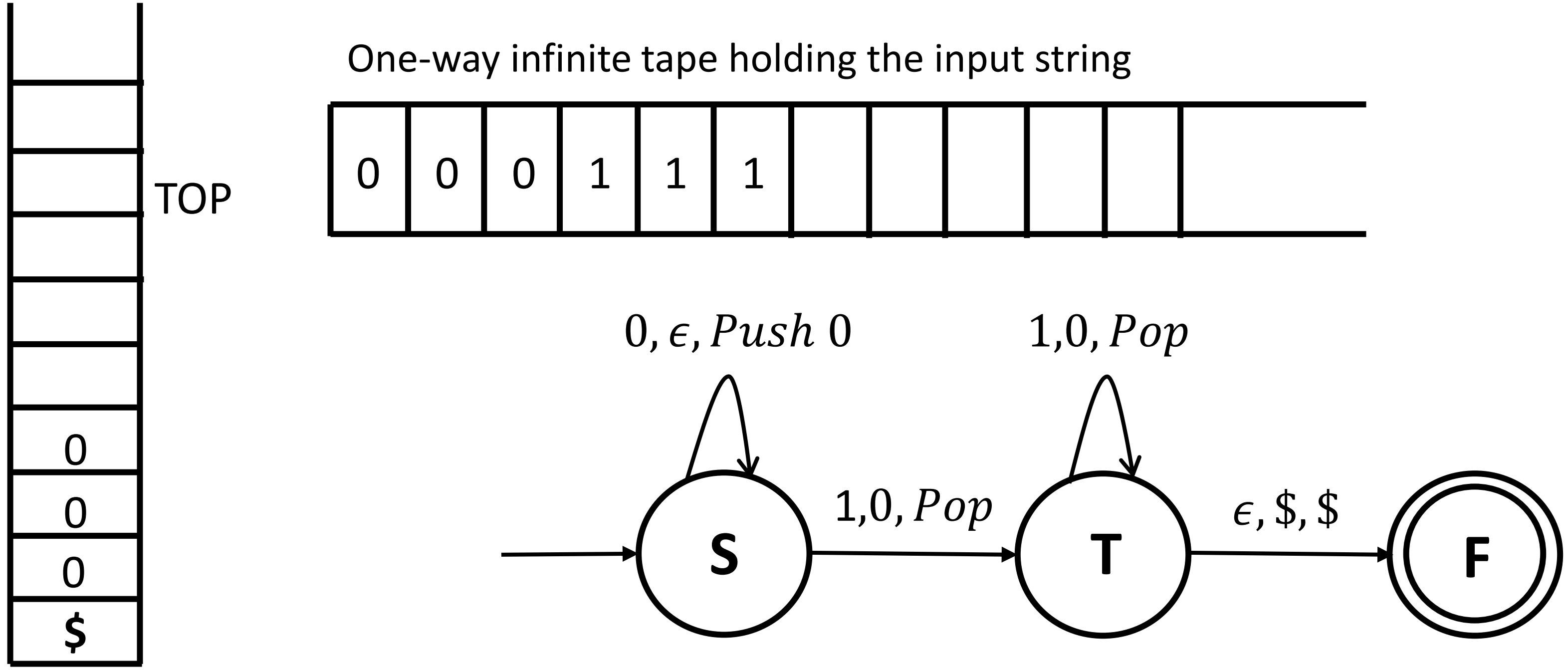


- A Pushdown Automata (PDA) is a finite automaton that has access to a stack.

- The FSM:
  - Transitions based on the Input symbol and the element at the top of the stack (e.g.: If I/P symbol = 0 & TOP = 0, transition from i to j) – **Pop 0**

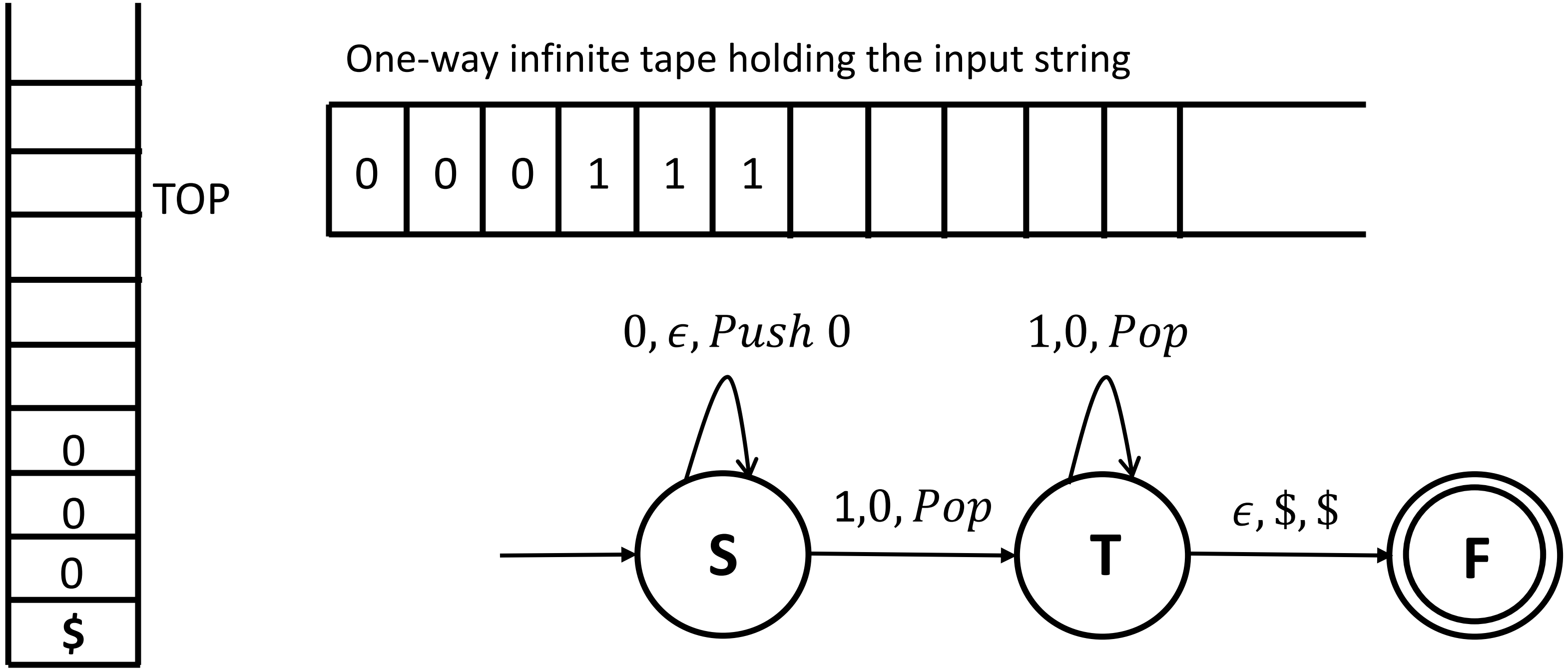
  - Pushes new elements into the Stack (e.g.: If I/P symbol = 0, PUSH 0, transition from i to j).

Pushdown Automata



- A Pushdown Automata (PDA) is a finite automaton that has access to a stack.
- The FSM:
  - Transitions based on the Input symbol and the element at the top of the stack
  - Pops the element at the top of the Stack.
  - Pushes new elements into the Stack.

Pushdown Automata

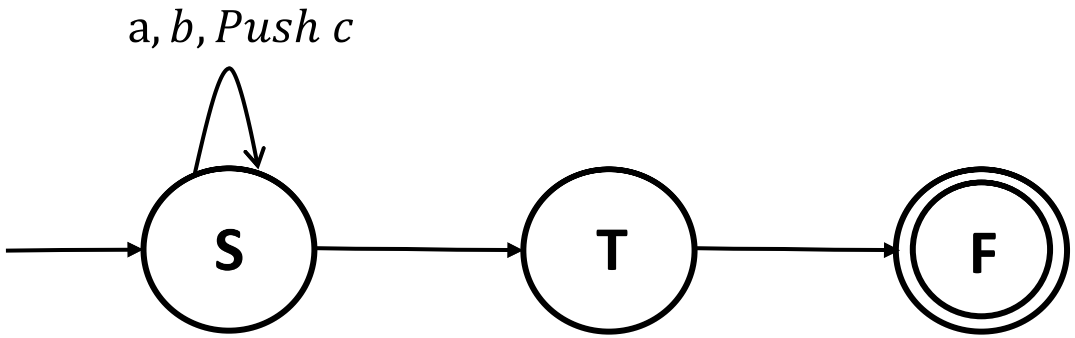


PDAs are **non-deterministic**.

- ϵ-transitions
- Multiple transitions/input symbol possible

Pushdown Automata

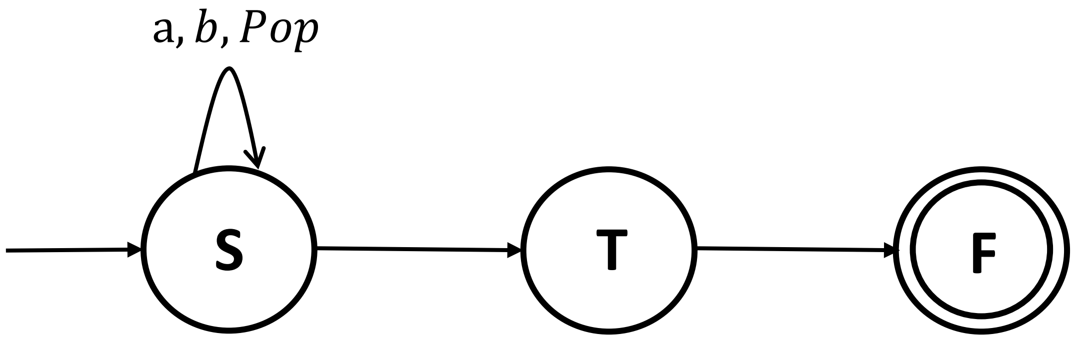
- How to represent a transition in a PDA?



If input symbol = a, Stack top = b (if b is popped) and Push c onto the Stack

Pushdown Automata

- How to represent a transition in a PDA?

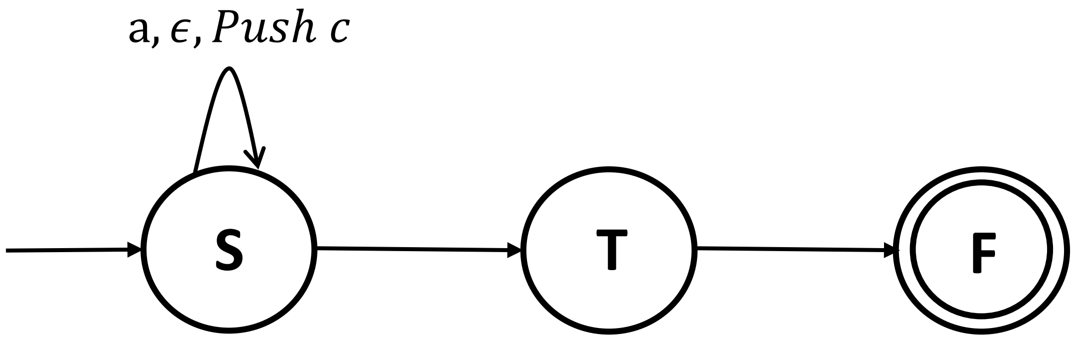


If input symbol = a, then Pop b

(If the symbol read is a and the stack TOP $= b$, then remain in S)

Pushdown Automata

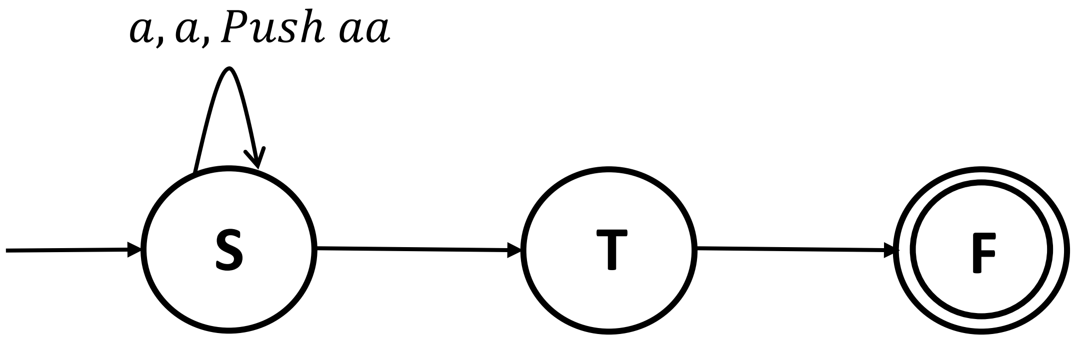
- How to represent a transition in a PDA?



If input symbol = a, then Push c

Pushdown Automata

- How to represent a transition in a PDA?



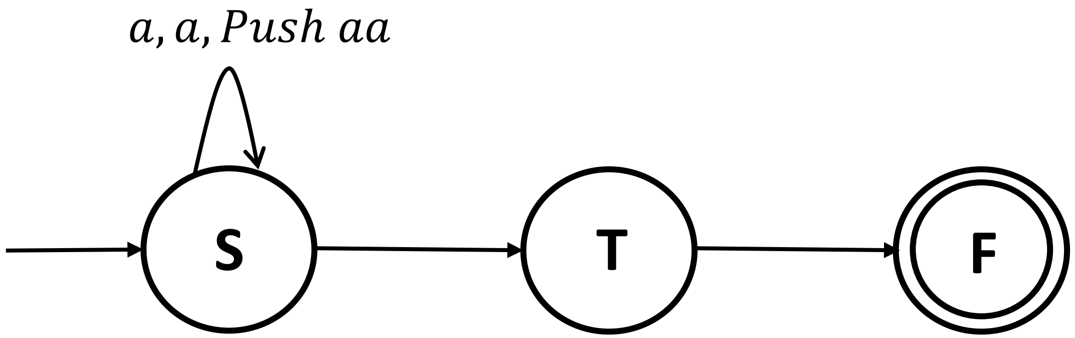
If input symbol = a, then Pop a and Push aa.

So effectively, the PDA pushes a onto the stack if it reads a on the input tape and the stack top = a.

This is a “shorthand” for describing two operations:

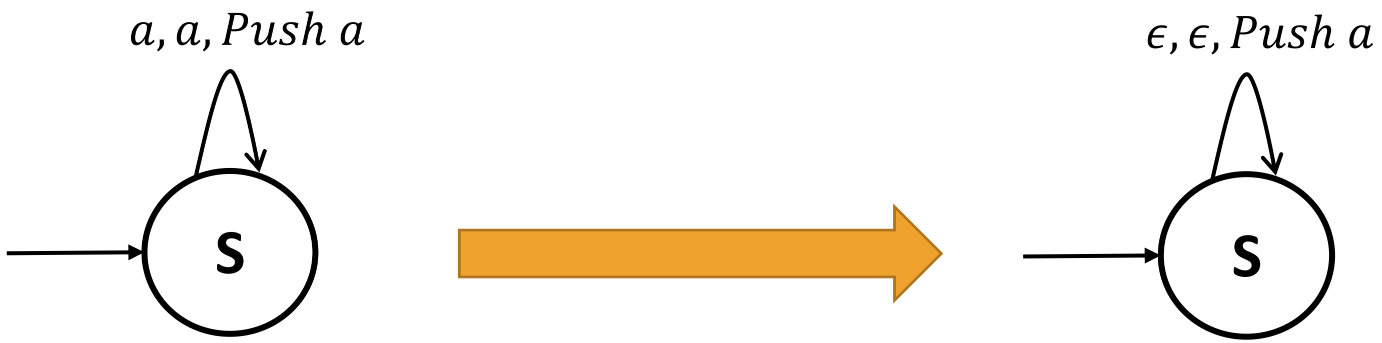
Pushdown Automata

- How to represent a transition in a PDA?



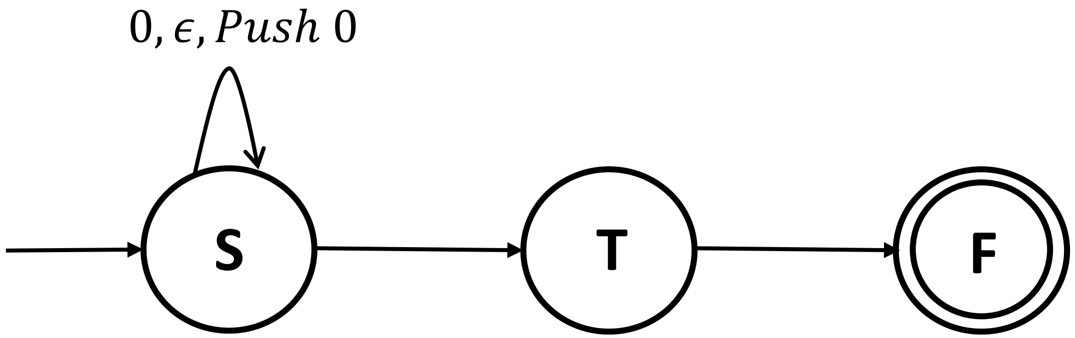
If input symbol = a, then Pop a and Push aa.

So effectively, the PDA pushes a onto the stack if it reads a on the input tape and the stack top = a.



Pushdown Automata

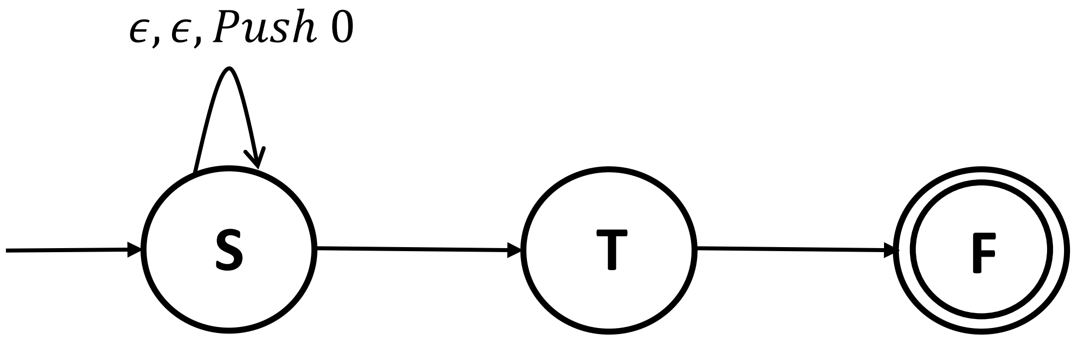
- How to represent a transition in a PDA?



If input symbol = 0, Push 0 onto the Stack irrespective of the element at the top of the stack

Pushdown Automata

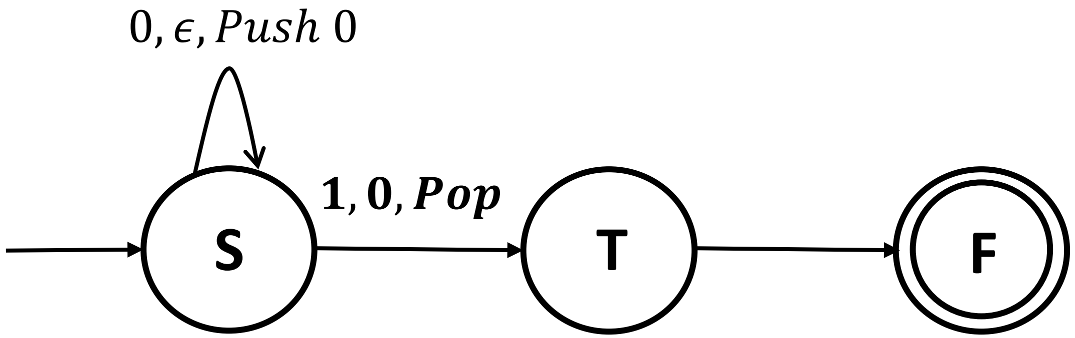
- How to represent a transition in a PDA?



Without reading the input symbol and the Stack top, Push 0 onto the Stack

Pushdown Automata

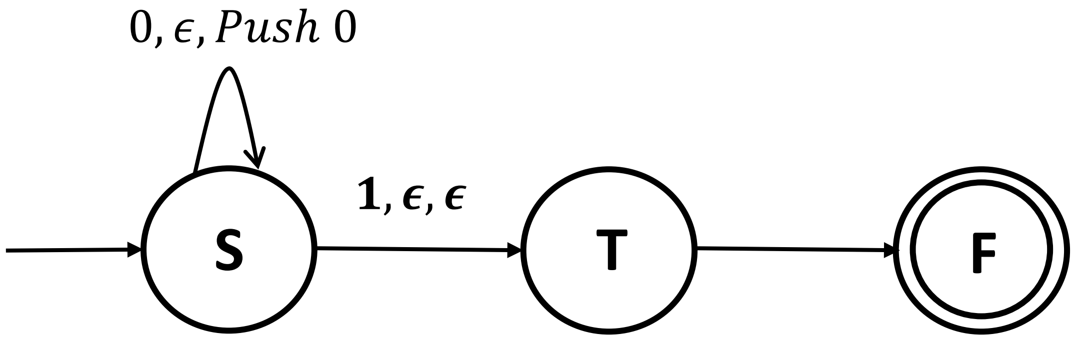
- How to represent a transition in a PDA?



If the input symbol is 1, and the element at the top of the stack is 0 **(Pop 0)**, then transition from S to T

Pushdown Automata

- How to represent a transition in a PDA?

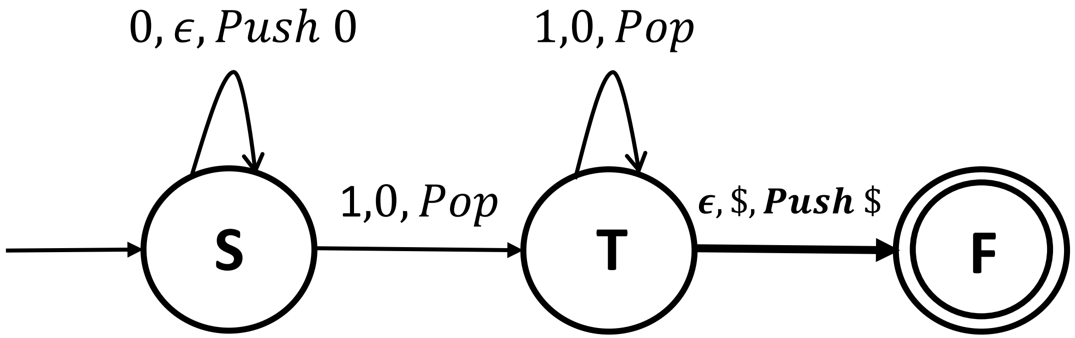


If the input symbol is 1, transition to T by ignoring the stack top completely.

If this happens at every step of the execution of the PDA, then it is as powerful as an NFA.

Pushdown Automata

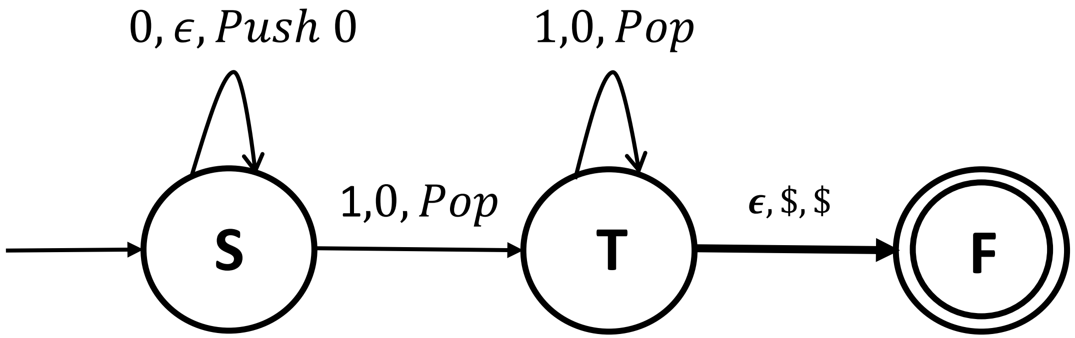
- How to represent a transition in a PDA?



If the Stack is empty, i.e. TOP = \$, transition to F from T, without reading the input

Pushdown Automata

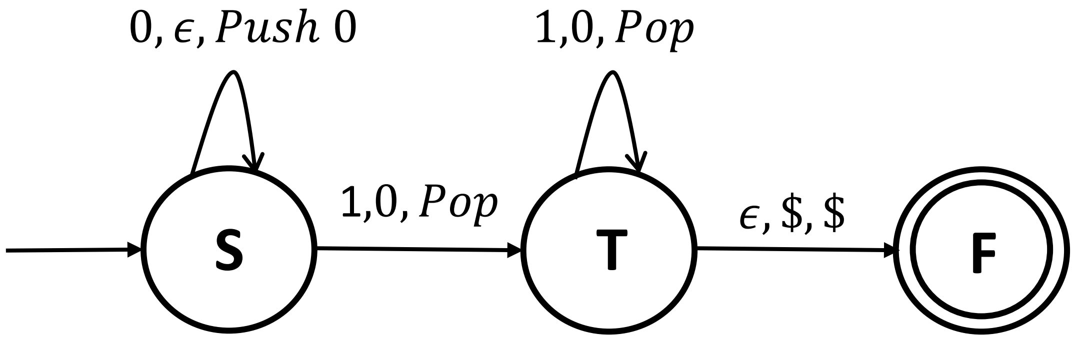
- How to represent a transition in a PDA?



If the Stack is empty, i.e. TOP = \$, transition to F from T, without reading the input

Pushdown Automata

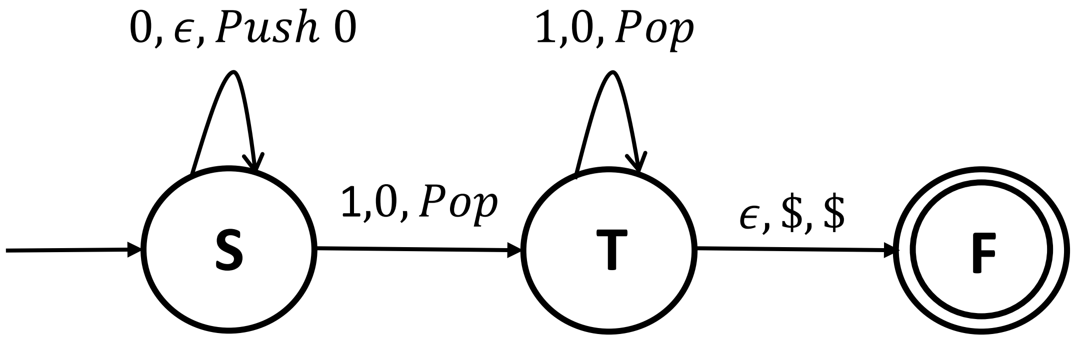
- How to represent a transition in a PDA?



What is the language accepted by this PDA?

Pushdown Automata

- How to represent a transition in a PDA?

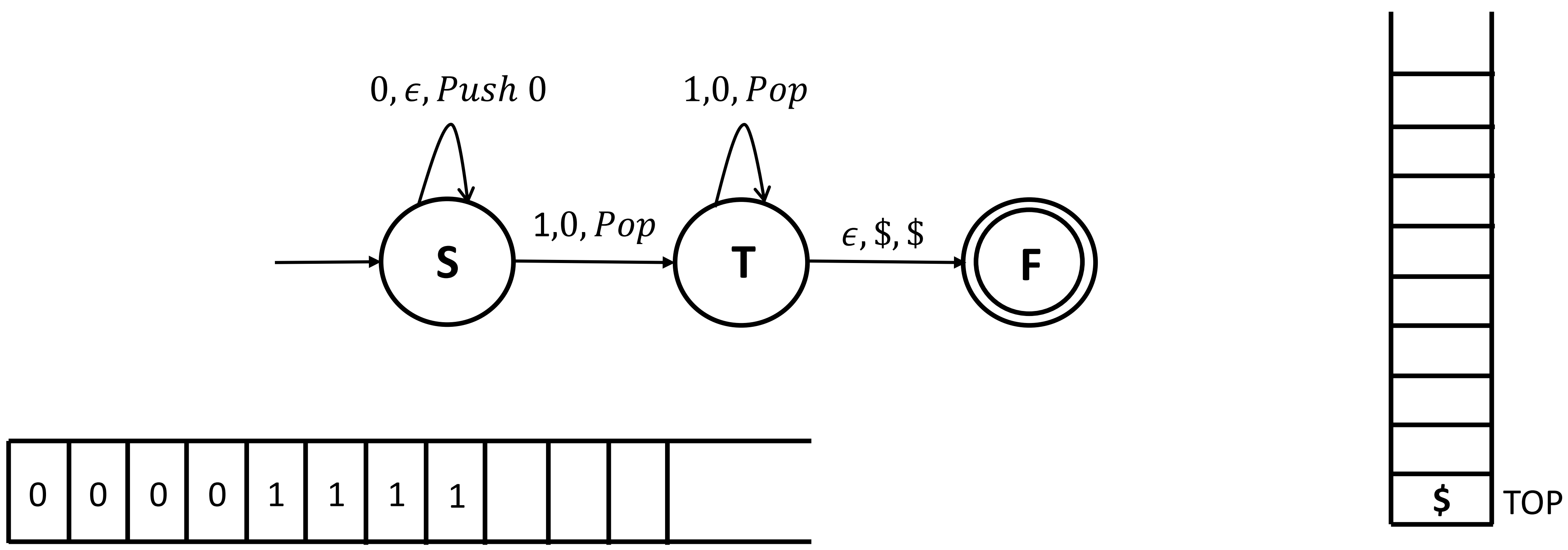


What is the language recognized by this PDA?

Verify that it is $\boldsymbol{L = \{0^n 1^n, n \geq 1\}}$

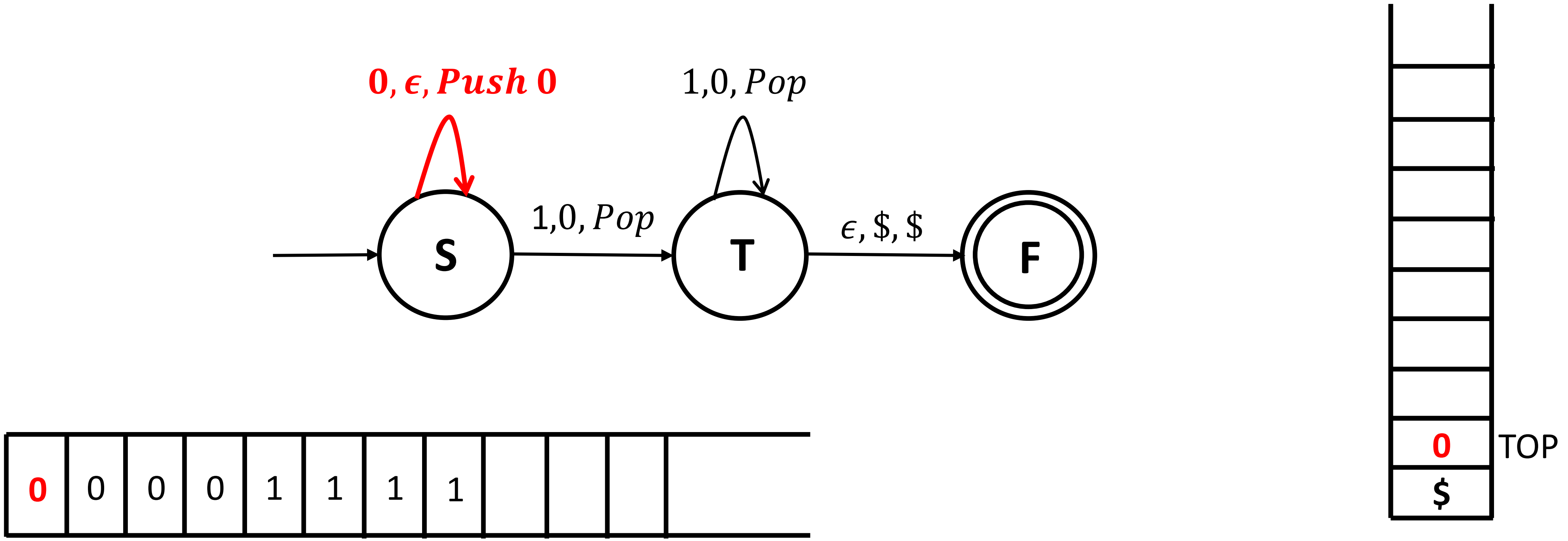
Pushdown Automata

What is the language recognized by this PDA?



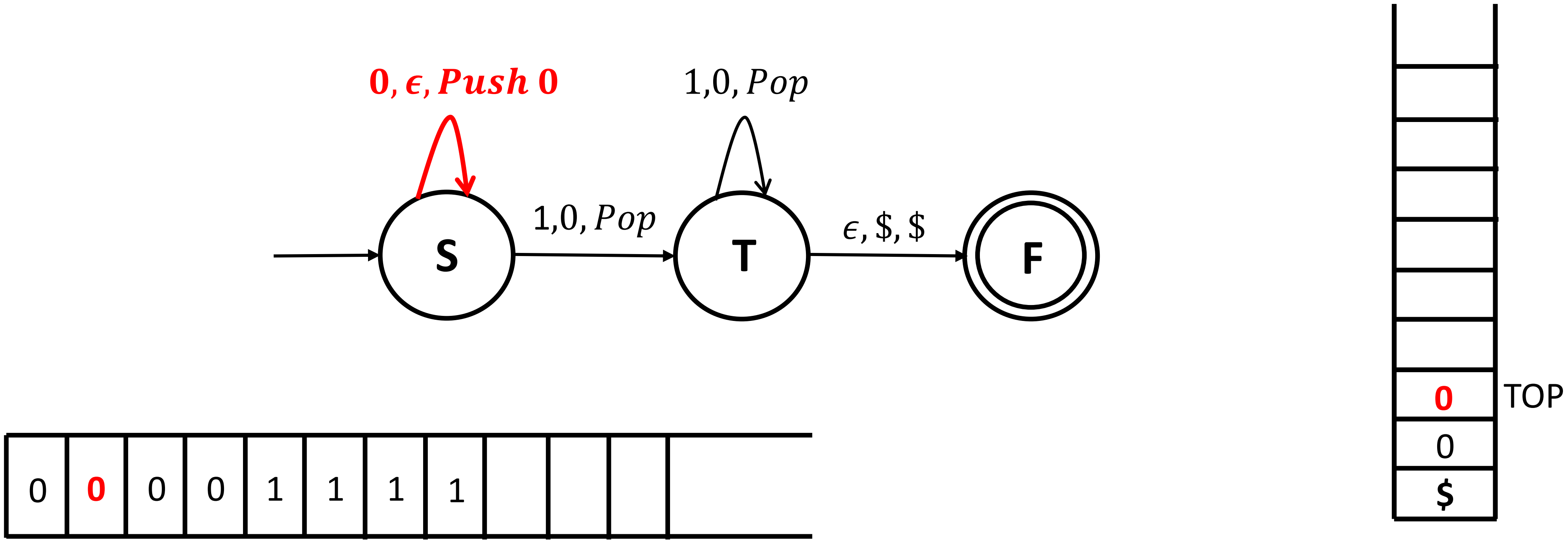
Pushdown Automata

What is the language recognized by this PDA?



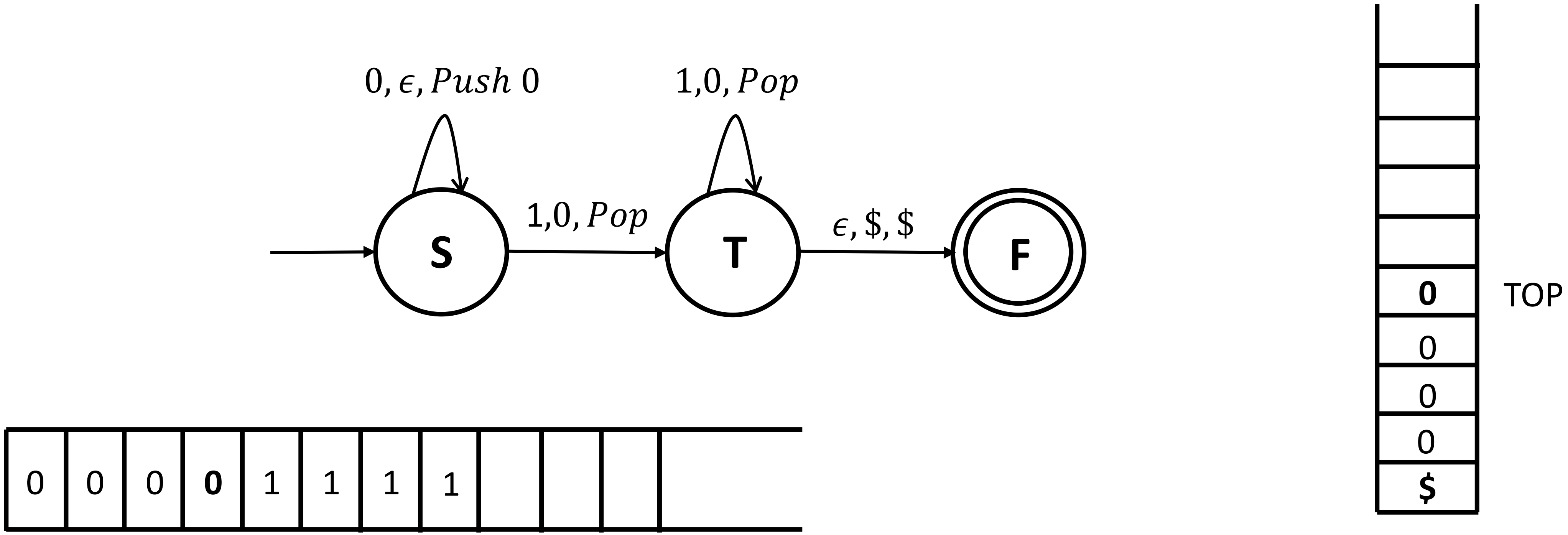
Pushdown Automata

What is the language recognized by this PDA?



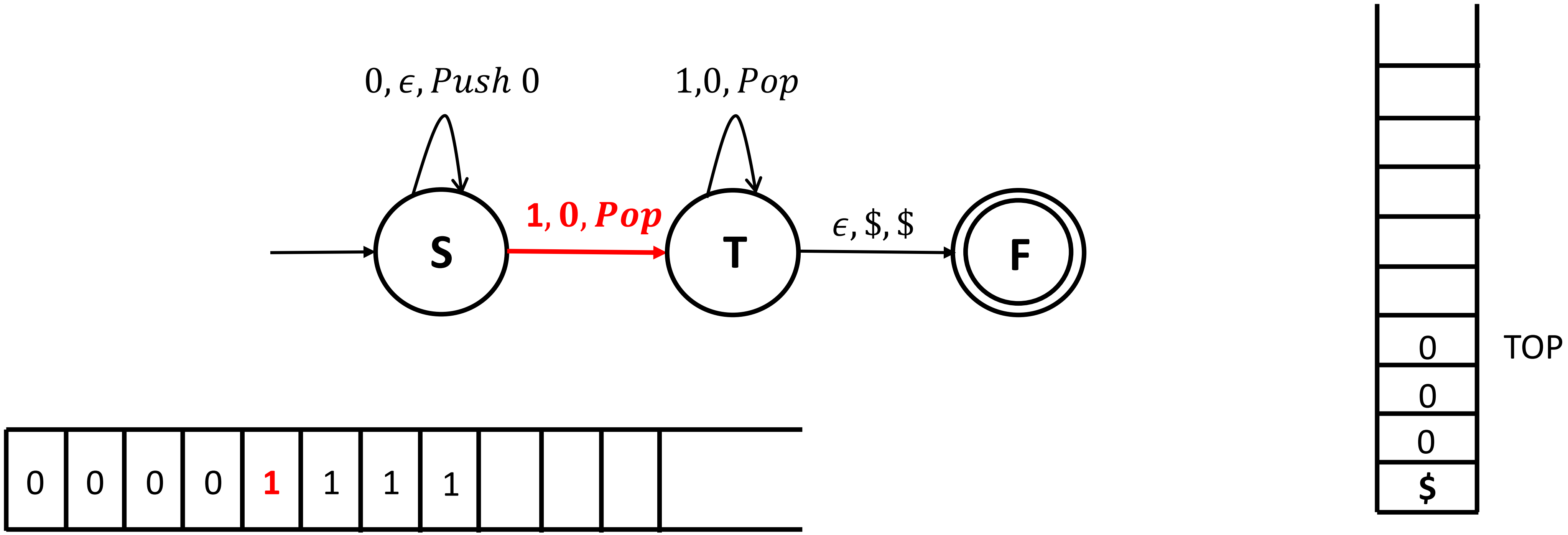
Pushdown Automata

What is the language recognized by this PDA?



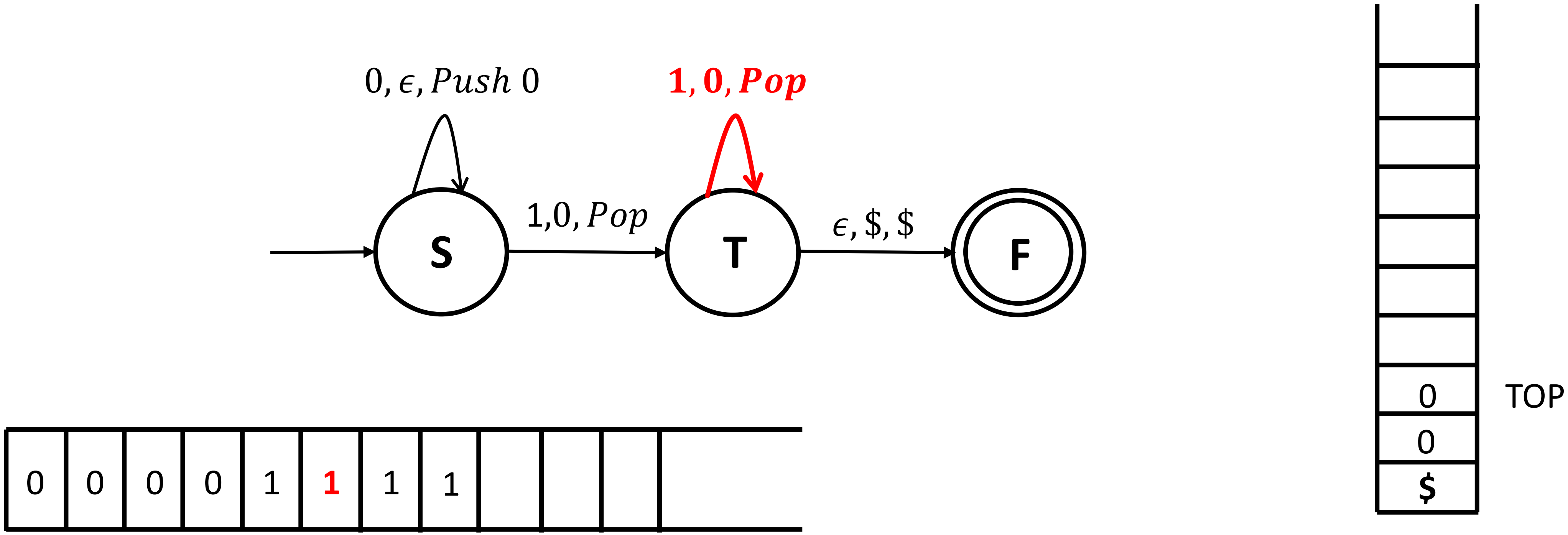
Pushdown Automata

What is the language recognized by this PDA?



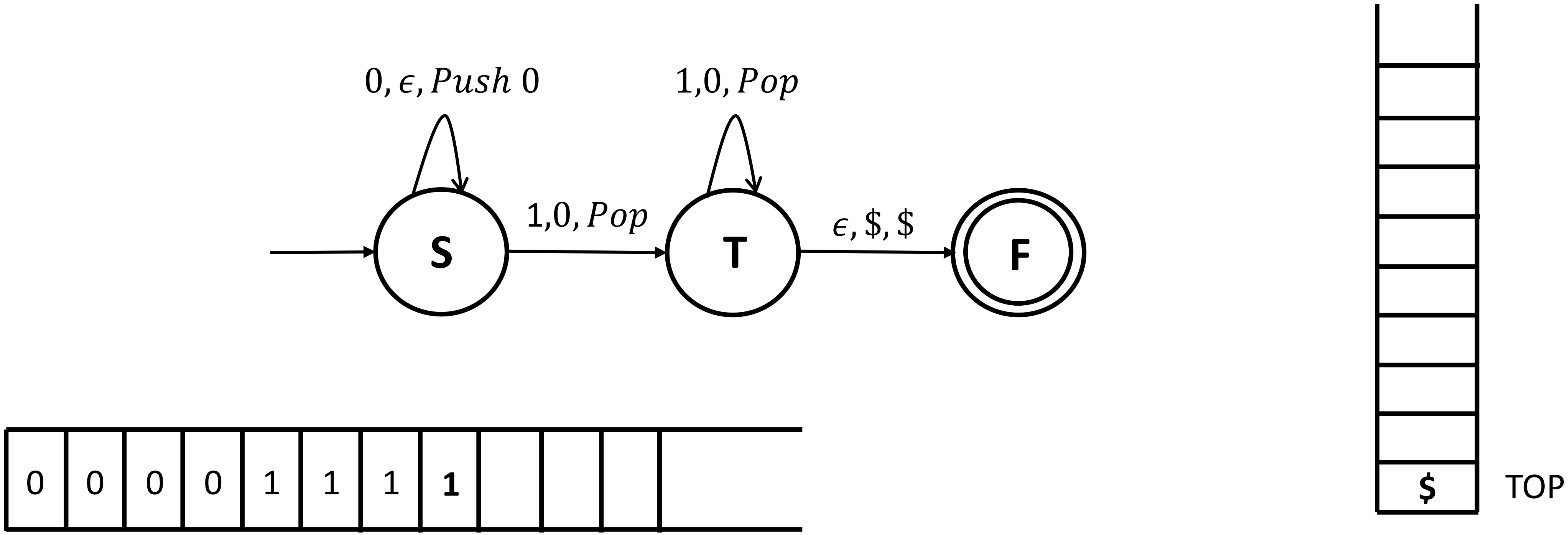
Pushdown Automata

What is the language recognized by this PDA?



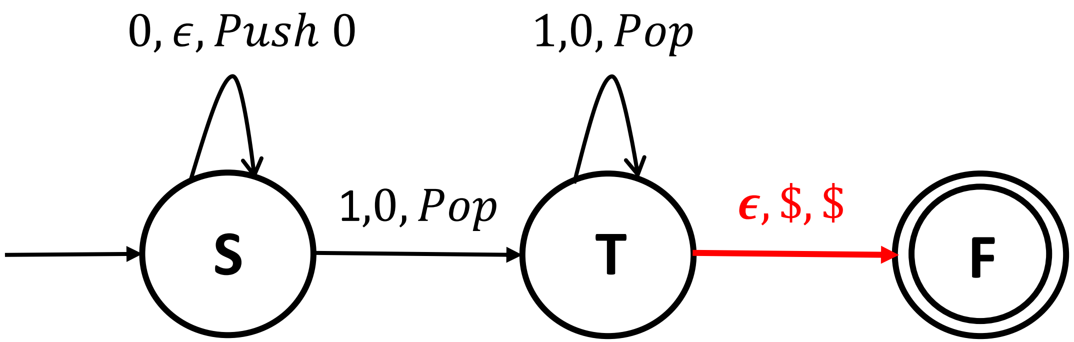
Pushdown Automata

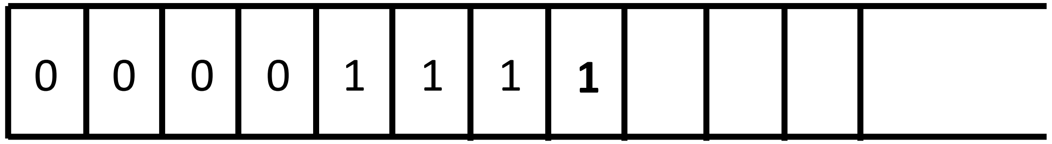
What is the language recognized by this PDA?

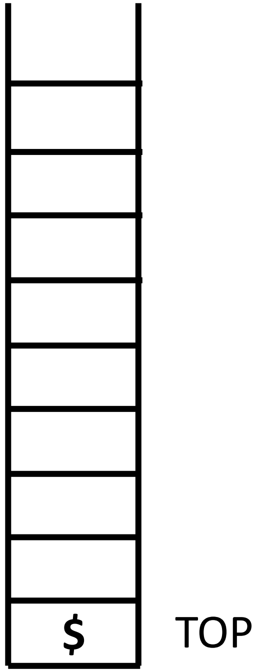


Pushdown Automata

What is the language recognized by this PDA?



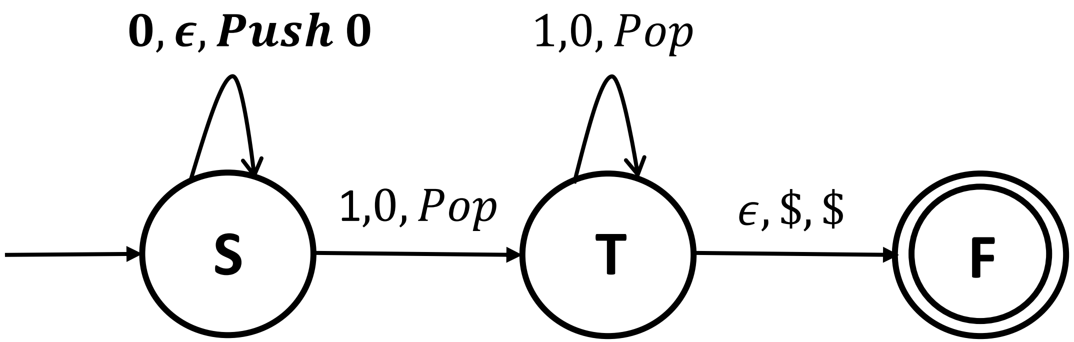




The language recognized by the PDA: $\boldsymbol{L = \{0^n 1^n, n \geq 1\}}$

Pushdown Automata

What is the language recognized by this PDA?

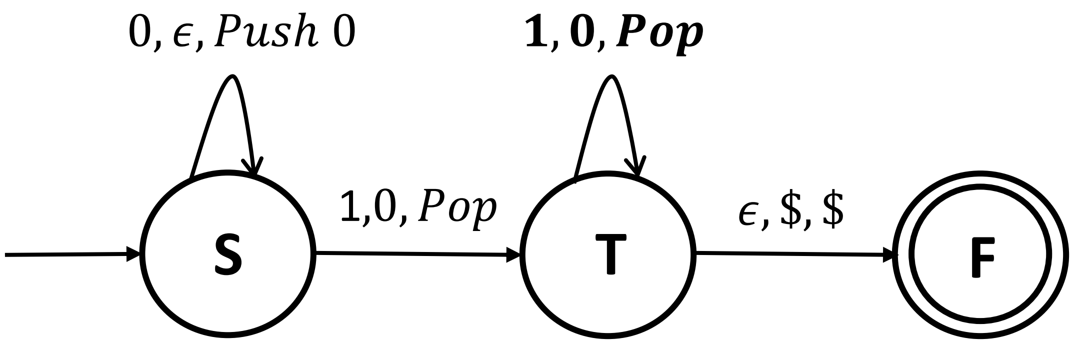


In some references (such as Sipser):

- The transitions of the PDA are labelled as "$\boldsymbol{a}, \boldsymbol{b} \rightarrow \boldsymbol{c}$", implying: If the input symbol read is $\boldsymbol{a}$, the element at the top of the stack is $\boldsymbol{b}$ (pop $\boldsymbol{b}$) and push $\boldsymbol{c}$ on to the Stack.

Pushdown Automata

What is the language recognized by this PDA?



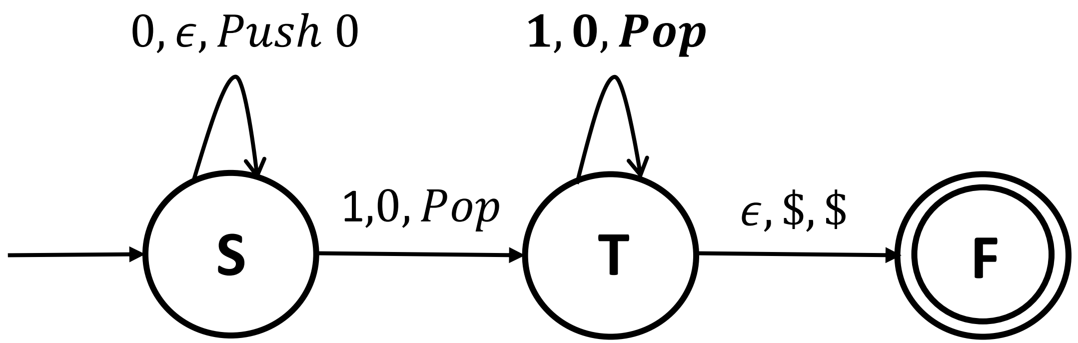
In some references (such as Sipser):

- The transitions of the PDA are labelled as "$\boldsymbol{a}, \boldsymbol{b} \rightarrow \boldsymbol{c}$", implying: If the input symbol read is $\boldsymbol{a}$, the element at the top of the stack is $\boldsymbol{b}$, then pop $\boldsymbol{b}$ and push $\boldsymbol{c}$ on to the Stack.

- The label "$\boldsymbol{a}, \boldsymbol{b} \rightarrow \boldsymbol{\epsilon}$" implies that if the input symbol is $\boldsymbol{a}$ then **pop b**.

Pushdown Automata

What is the language recognized by this PDA?



In some references (such as Sipser):

- The transitions of the PDA are labelled as "$\boldsymbol{a, b \rightarrow c}$", implying: If the input symbol read is $\boldsymbol{a}$, the element at the top of the stack is $\boldsymbol{b}$, then pop $\boldsymbol{b}$ and push $\boldsymbol{c}$ on to the Stack.

- The label "$\boldsymbol{a, b \rightarrow \epsilon}$" implies that if the input symbol is $\boldsymbol{a}$ and $\boldsymbol{b}$ is **popped**.

- The symbol signifying the bottom of the Stack \$ is pushed at the very beginning.

Pushdown Automata

Formally, a PDA M is a 6-tuple $(\mathrm{Q}, \Sigma, \Gamma, \delta, q_0, F\ )$ where

- Q is a finite set called the ***states.***
- Σ is the set of input ***alphabets.***
- Γ is the set of ***Stack alphabets***
- $\delta: Q \times \Sigma_\epsilon \times \Gamma_\epsilon \mapsto \mathcal{P}(Q \times \Gamma_\epsilon)$ is the ***transition function*** [$\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$ and $\Gamma_\epsilon\ = \Gamma \cup \{\epsilon\}$]
- $q_0 \in Q$ is the ***start state***.
- $F \subseteq Q$ is the set of ***accepting states***.

Pushdown Automata

Formally, a PDA M is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F\ )$ where

- Q is a finite set called the ***states.***
- Σ is the set of input ***alphabets.***
- Γ is the set of ***Stack alphabets***
- $\delta: Q \times \Sigma_\epsilon \times \Gamma_\epsilon \mapsto \mathcal{P}(Q \times \Gamma_\epsilon)$ is the ***transition function*** [$\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$ and $\Gamma_\epsilon = \Gamma \cup \{\epsilon\}$]
- $q_0 \in Q$ is the ***start state***.
- $F \subseteq Q$ is the set of ***accepting states***.

A PDA accepts a string $w \in L$, if there exists a run such that

- It **reaches a final state** when the entire string is read.

OR

- The **stack is empty** when the entire string is read.

Pushdown Automata

Formally, a PDA M is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where

- Q is a finite set called the ***states.***
- Σ is the set of input ***alphabets.***
- Γ is the set of ***Stack alphabets***
- $\delta: Q \times \Sigma_\epsilon \times \Gamma_\epsilon \mapsto \mathcal{P}(Q \times \Gamma_\epsilon)$ is the ***transition function***
- $q_0 \in Q$ is the ***start state***.
- $F \subseteq Q$ is the set of ***accepting states***.

[$\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$ and $\Gamma_\epsilon = \Gamma \cup \{\epsilon\}$]

A PDA accepts a string $w \in L$, if there exists a run such that

- It **reaches a final state** when the entire string is read.

OR

- The **stack is empty** when the entire string is read.

These two notions of acceptance are equivalent

Pushdown Automata

Formally, a PDA M is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F\ )$ where

- Q is a finite set called the ***states.***
- Σ is the set of input ***alphabets.***
- Γ is the set of ***Stack alphabets***
- $\delta: Q \times \Sigma_\epsilon \times \Gamma_\epsilon \mapsto \mathcal{P}(Q \times \Gamma_\epsilon)$ is the ***transition function*** [$\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$ and $\Gamma_\epsilon = \Gamma \cup \{\epsilon\}$]
- $q_0 \in Q$ is the ***start state***.
- $F \subseteq Q$ is the set of ***accepting states***.

Transition function:

- $\boldsymbol{\delta(q_i, a, b) = (q_j, c)}$: If the input symbol read is a and b is popped, then push c onto the stack and transition from q_i to q_j

Pushdown Automata

Formally, a PDA M is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where

- Q is a finite set called the ***states.***
- Σ is the set of input ***alphabets.***
- Γ is the set of ***Stack alphabets***
- $\delta: Q \times \Sigma_\epsilon \times \Gamma_\epsilon \mapsto \mathcal{P}(Q \times \Gamma_\epsilon)$ is the ***transition function*** [$\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$ and $\Gamma_\epsilon = \Gamma \cup \{\epsilon\}$]
- $q_0 \in Q$ is the ***start state***.
- $F \subseteq Q$ is the set of ***accepting states***.

Transition function:

- $\boldsymbol{\delta(q_i, a, b) = (q_j, c)}$: If the input symbol read is a and b is popped, then push c onto the stack and transition from q_i to q_j

- $\boldsymbol{\delta(q_i, a, \epsilon) = (q_j, c)}$:

Pushdown Automata

Formally, a PDA M is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F\ )$ where

- Q is a finite set called the ***states.***
- Σ is the set of input ***alphabets.***
- Γ is the set of ***Stack alphabets***
- $\delta: Q \times \Sigma_\epsilon \times \Gamma_\epsilon \mapsto \mathcal{P}(Q \times \Gamma_\epsilon)$ is the ***transition function*** [$\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$ and $\Gamma_\epsilon = \Gamma \cup \{\epsilon\}$]
- $q_0 \in Q$ is the ***start state***.
- $F \subseteq Q$ is the set of ***accepting states***.

Transition function:

- $\boldsymbol{\delta(q_i, a, b) = (q_j, c)}$: If the input symbol read is a and b is popped, then push c onto the stack and transition from q_i to q_j

- $\boldsymbol{\delta(q_i, a, \epsilon) = (q_j, c)}$: If the input symbol read is a, then push c onto the stack and transition from q_i to q_j

Pushdown Automata

Formally, a PDA M is a 6-tuple $(\mathrm{Q}, \Sigma, \Gamma, \delta, q_0, F)$ where

- Q is a finite set called the ***states.***
- Σ is the set of input ***alphabets.***
- Γ is the set of ***Stack alphabets***
- $\delta: Q \times \Sigma_\epsilon \times \Gamma_\epsilon \mapsto \mathcal{P}(Q \times \Gamma_\epsilon)$ is the ***transition function*** [$\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$ and $\Gamma_\epsilon = \Gamma \cup \{\epsilon\}$]
- $q_0 \in Q$ is the ***start state***.
- $F \subseteq Q$ is the set of ***accepting states***.

Transition function:

- $\boldsymbol{\delta(q_i, a, b) = (q_j, c)}$: If the input symbol read is a and b is popped, then push c onto the stack and transition from q_i to q_j

- $\boldsymbol{\delta(q_i, a, \epsilon) = (q_j, c)}$: If the input symbol read is a, then push c onto the stack and transition from q_i to q_j

- $\boldsymbol{\delta(q_i, a, b) = (q_j, \epsilon)}$:

Pushdown Automata

Formally, a PDA M is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F\ )$ where

- Q is a finite set called the ***states.***
- Σ is the set of input ***alphabets.***
- Γ is the set of ***Stack alphabets***
- $\delta: Q \times \Sigma_\epsilon \times \Gamma_\epsilon \mapsto \mathcal{P}(Q \times \Gamma_\epsilon)$ is the ***transition function*** [$\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$ and $\Gamma_\epsilon = \Gamma \cup \{\epsilon\}$]
- $q_0 \in Q$ is the ***start state***.
- $F \subseteq Q$ is the set of ***accepting states***.

Transition function:

- $\boldsymbol{\delta(q_i, a, b) = (q_j, c)}$: If the input symbol read is a and b is popped, then push c onto the stack and transition from q_i to q_j

- $\boldsymbol{\delta(q_i, a, \epsilon) = (q_j, c)}$: If the input symbol read is a, then push c onto the stack and transition from q_i to q_j

- $\boldsymbol{\delta(q_i, a, b) = (q_j, \epsilon)}$: If the input symbol read is a, and the stack top = b (b is popped) then transition from q_i to q_j

- $\boldsymbol{\delta(q_i, \epsilon, \$) = (q_j, \$)}$:

Pushdown Automata

Formally, a PDA M is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where

- Q is a finite set called the ***states.***
- Σ is the set of input ***alphabets.***
- Γ is the set of ***Stack alphabets***
- $\delta: Q \times \Sigma_\epsilon \times \Gamma_\epsilon \mapsto \mathcal{P}(Q \times \Gamma_\epsilon)$ is the ***transition function*** [$\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$ and $\Gamma_\epsilon = \Gamma \cup \{\epsilon\}$]
- $q_0 \in Q$ is the ***start state***.
- $F \subseteq Q$ is the set of ***accepting states***.

Transition function:

- $\boldsymbol{\delta(q_i, a, b) = (q_j, c)}$: If the input symbol read is a and b is popped, then push c onto the stack and transition from q_i to q_j

- $\boldsymbol{\delta(q_i, a, \epsilon) = (q_j, c)}$: If the input symbol read is a, then push c onto the stack and transition from q_i to q_j

- $\boldsymbol{\delta(q_i, a, b) = (q_j, \epsilon)}$: If the input symbol read is a, and the stack top = b, then pop b and transition from q_i to q_j

- $\boldsymbol{\delta(q_i, \epsilon, \$) = (q_j, \$)}$: Transition from q_i to q_j if the stack is empty.

Pushdown Automata

Formally, a PDA M is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where

- Q is a finite set called the ***states.***
- Σ is the set of input ***alphabets.***
- Γ is the set of ***Stack alphabets***
- $\delta: Q \times \Sigma_\epsilon \times \Gamma_\epsilon \mapsto \mathcal{P}(Q \times \Gamma_\epsilon)$ is the ***transition function*** [$\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$ and $\Gamma_\epsilon = \Gamma \cup \{\epsilon\}$]
- $q_0 \in Q$ is the ***start state***.
- $F \subseteq Q$ is the set of ***accepting states***.

Transition function:

- $\boldsymbol{\delta(q_i, a, b) = (q_j, c)}$: If the input symbol read is a and b is popped, then push c onto the stack and transition from q_i to q_j

- If the input symbol read is a and a is popped, then Push a and remain at q_i : ?

Pushdown Automata

Formally, a PDA M is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where

- Q is a finite set called the ***states.***
- Σ is the set of input ***alphabets.***
- Γ is the set of ***Stack alphabets***
- $\delta: Q \times \Sigma_\epsilon \times \Gamma_\epsilon \mapsto \mathcal{P}(Q \times \Gamma_\epsilon)$ is the ***transition function*** [$\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$ and $\Gamma_\epsilon = \Gamma \cup \{\epsilon\}$]
- $q_0 \in Q$ is the ***start state***.
- $F \subseteq Q$ is the set of ***accepting states***.

Transition function:

- $\boldsymbol{\delta(q_i, a, b) = (q_j, c)}$: If the input symbol read is a and b is popped, then push c onto the stack and transition from q_i to q_j

- If the input symbol read is a and a is popped, then Push a and remain at q_i : $\boldsymbol{\delta(q_i, a, a) = (q_i, a)}$

Pushdown Automata

Formally, a PDA M is a 6-tuple $(\mathrm{Q}, \Sigma, \Gamma, \delta, q_0, F)$ where

- Q is a finite set called the ***states.***
- Σ is the set of input ***alphabets.***
- Γ is the set of ***Stack alphabets***
- $\delta: Q \times \Sigma_\epsilon \times \Gamma_\epsilon \mapsto \mathcal{P}(Q \times \Gamma_\epsilon)$ is the ***transition function*** $[\,\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$ and $\Gamma_\epsilon = \Gamma \cup \{\epsilon\}\,]$
- $q_0 \in Q$ is the ***start state***.
- $F \subseteq Q$ is the set of ***accepting states***.

- The Language of the PDA P is the set of strings the PDA accepts, i.e.

$$L = \{w | P \text{ accepts } w\}$$

There exists an accepting run for $\boldsymbol{w}$ on $\boldsymbol{P}$

- If $\mathcal{L}(P) = L$, then the PDA P recognizes L

Pushdown Automata

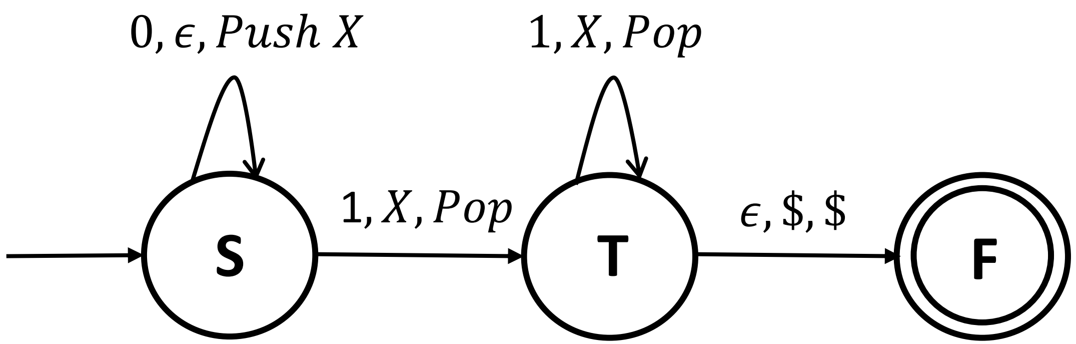
Formally, a PDA M is a 6-tuple $(\mathrm{Q}, \Sigma, \Gamma, \delta, q_0, F)$ where

- Q is a finite set called the ***states.***
- Σ is the set of input ***alphabets.***
- Γ is the set of ***Stack alphabets***
- $\delta: Q \times \Sigma_\epsilon \times \Gamma_\epsilon \mapsto \mathcal{P}(Q \times \Gamma_\epsilon)$ is the ***transition function*** $[\ \Sigma_\epsilon = \Sigma \cup \{\epsilon\}$ and $\Gamma_\epsilon = \Gamma \cup \{\epsilon\}\ ]$
- $q_0 \in Q$ is the ***start state***.
- $F \subseteq Q$ is the set of ***accepting states***.

- The Language of the PDA P is the set of strings the PDA accepts, i.e.

$$L = \{w | P \text{ accepts } w\}$$

- If $\mathcal{L}(P) = L$, then the PDA P recognizes L
- Stack alphabet **can be different** from the input alphabet



Pushdown Automata

Formally, a PDA M is a 6-tuple $(\mathrm{Q}, \Sigma, \Gamma, \delta, q_0, F)$ where

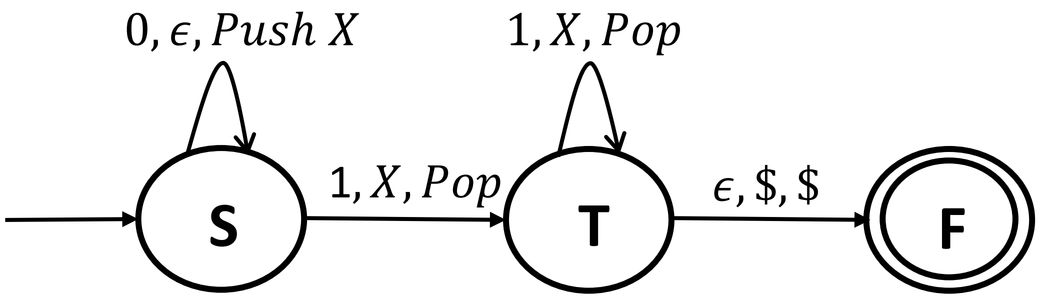
- Q is a finite set called the ***states.***
- Σ is the set of input ***alphabets.***
- Γ is the set of ***Stack alphabets***
- $\delta: Q \times \Sigma_\epsilon \times \Gamma_\epsilon \mapsto \mathcal{P}(Q \times \Gamma_\epsilon)$ is the ***transition function*** $[\ \Sigma_\epsilon = \Sigma \cup \{\epsilon\}$ and $\Gamma_\epsilon = \Gamma \cup \{\epsilon\}\ ]$
- $q_0 \in Q$ is the ***start state***.
- $F \subseteq Q$ is the set of ***accepting states***.

- The Language of the PDA P is the set of strings the PDA accepts, i.e.

$$L = \{w | P \text{ accepts } w\}$$

- If $\mathcal{L}(P) = L$, then the PDA P recognizes L

- Stack alphabet **can be different** from the input alphabet



$\delta(S, 0, \epsilon) = (S, X)$

Pushdown Automata

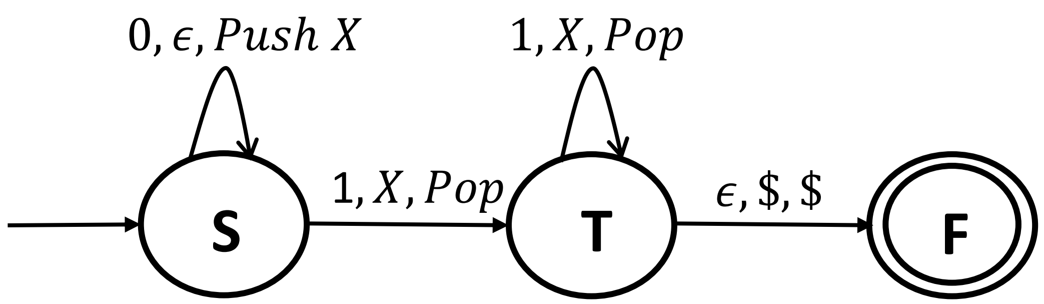
Formally, a PDA M is a 6-tuple $(\mathrm{Q}, \Sigma, \Gamma, \delta, q_0, F)$ where

- Q is a finite set called the ***states.***
- Σ is the set of input ***alphabets.***
- Γ is the set of ***Stack alphabets***
- $\delta: Q \times \Sigma_\epsilon \times \Gamma_\epsilon \mapsto \mathcal{P}(Q \times \Gamma_\epsilon)$ is the ***transition function*** $[\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$ and $\Gamma_\epsilon = \Gamma \cup \{\epsilon\}]$
- $q_0 \in Q$ is the ***start state***.
- $F \subseteq Q$ is the set of ***accepting states***.

- The Language of the PDA P is the set of strings the PDA accepts, i.e.

$$L = \{w | P \text{ accepts } w\}$$

- If $\mathcal{L}(P) = L$, then the PDA P recognizes L
- Stack alphabet **can be different** from the input alphabet



$$\delta(S, 0, \epsilon) = (S, X)$$
$$\delta(S, 1, X) = (T, \epsilon)$$

Pushdown Automata

Formally, a PDA M is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where

- Q is a finite set called the ***states.***
- Σ is the set of input ***alphabets.***
- Γ is the set of ***Stack alphabets***
- $\delta: Q \times \Sigma_\epsilon \times \Gamma_\epsilon \mapsto \mathcal{P}(Q \times \Gamma_\epsilon)$ is the ***transition function***

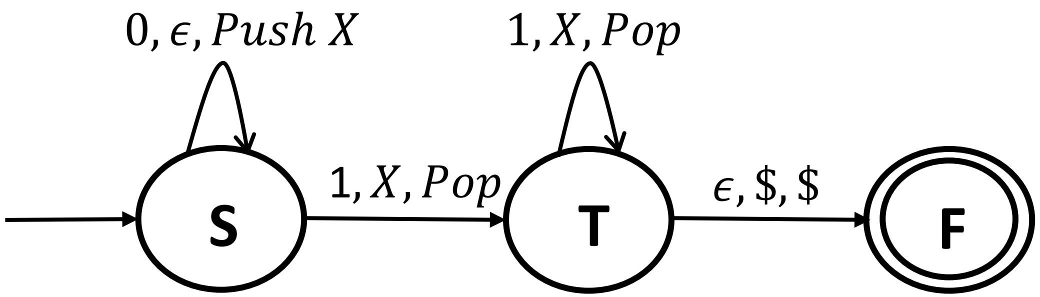
[$\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$ and $\Gamma_\epsilon = \Gamma \cup \{\epsilon\}$]

- $q_0 \in Q$ is the ***start state***.
- $F \subseteq Q$ is the set of ***accepting states***.

- The Language of the PDA P is the set of strings the PDA accepts, i.e.

$$L = \{w | P \text{ accepts } w\}$$

- If $\mathcal{L}(P) = L$, then the PDA P recognizes L
- Stack alphabet **can be different** from the input alphabet



$$\delta(S, 0, \epsilon) = (S, X)$$
$$\delta(S, 1, X) = (T, \epsilon)$$
$$\delta(T, 1, X) = (T, \epsilon)$$
$$\delta(T, \epsilon, \$) = (F, \$)$$

Pushdown Automata

Let $\Sigma = \{0,1\}$ consider the language $\boldsymbol{L} = \{\boldsymbol{w} \in \boldsymbol{\Sigma}^* \ |\boldsymbol{w}$ **is a Palindrome**$\}$. Design a PDA P that recognizes L.

Pushdown Automata

Let $\Sigma = \{0,1\}$ consider the language $\boldsymbol{L} = \{\boldsymbol{w} \in \boldsymbol{\Sigma}^* \ |\boldsymbol{w}$ **is a Palindrome}**. Design a PDA P that recognizes L.

Intuition

- Push first half of the input string onto the stack.
- Verify that the second half of the symbols match the first half: Keep Popping the stack until the end of the input.

Pushdown Automata

Let $\Sigma = \{0,1\}$ consider the language $\boldsymbol{L} = \{\boldsymbol{w} \in \boldsymbol{\Sigma}^* \mid \boldsymbol{w}$ **is a Palindrome}**. Design a PDA P that recognizes L.

Intuition

- Push first half of the input string onto the stack.
- Verify that the second half of the symbols match the first half: Keep Popping the stack until the end of the input.

- How can the PDA know that the middle of the input has been reached.
    - The PDA does this non-deterministically (by taking ϵ transitions).

Pushdown Automata

Let $\Sigma = \{0,1\}$ consider the language $\boldsymbol{L} = \{\boldsymbol{w} \in \boldsymbol{\Sigma}^* \ |\boldsymbol{w}$ **is a Palindrome}**. Design a PDA P that recognizes L.

Intuition

- Push first half of the input string onto the stack.
- Verify that the second half of the symbols match the first half: Keep Popping the stack until the end of the input.

- How can the PDA know that the middle of the input has been reached.
  - The PDA does this non-deterministically (by taking ϵ transitions).

- The above intuition is applicable for even length palindromes of the form ww^R.

- What about odd length palindromes?
  - Non-determinism to the rescue once again

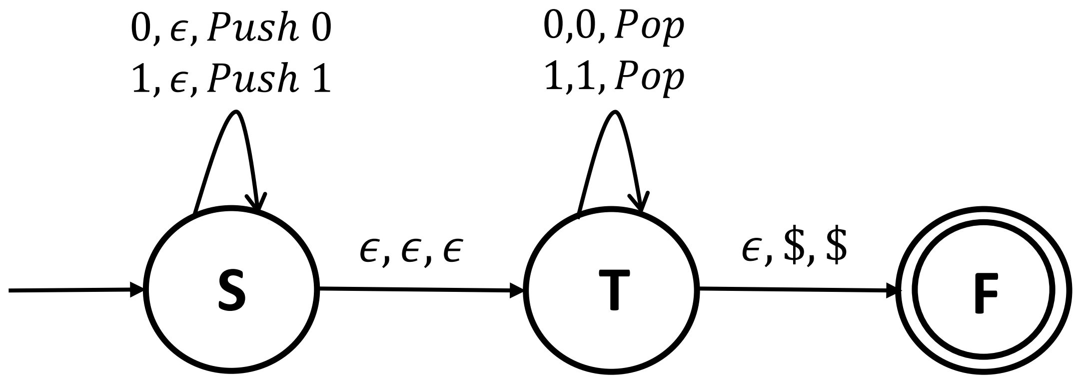
Pushdown Automata

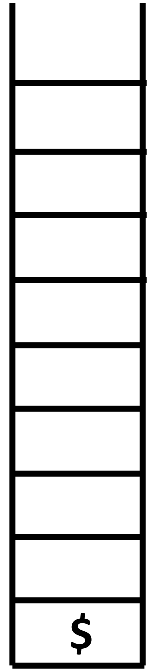
Let $\Sigma = \{0,1\}$ consider the language $\boldsymbol{L} = \{\boldsymbol{w} \in \boldsymbol{\Sigma}^* \mid \boldsymbol{w}$ **is a Palindrome}**. Design a PDA P that recognizes L.

Intuition

- Push first half of the input string onto the stack.
- Verify that the second half of the symbols match the first half: Keep Popping the stack until the end of the input.

- How can the PDA know that the middle of the input has been reached.
  - The PDA does this non-deterministically (by taking ϵ transitions).





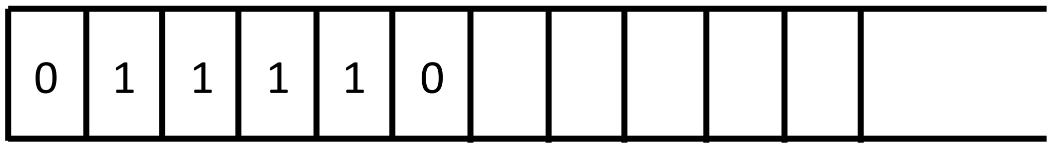
Pushdown Automata

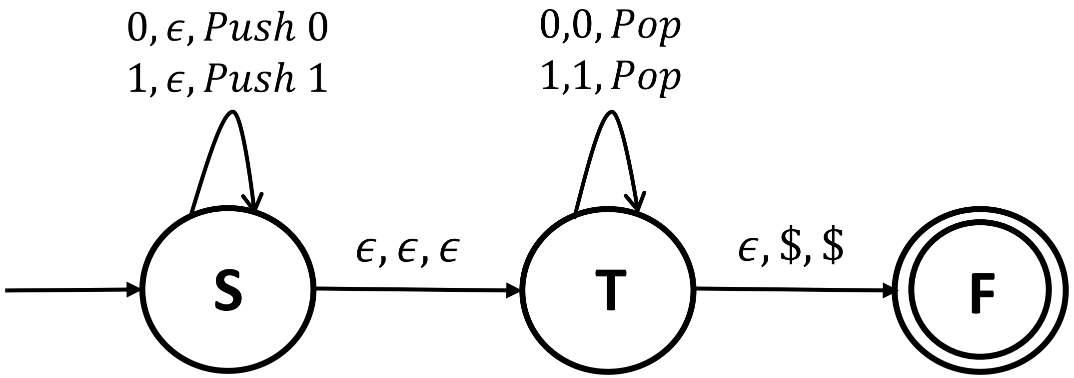
Let $\Sigma = \{0,1\}$ consider the language $\boldsymbol{L} = \{\boldsymbol{w} \in \boldsymbol{\Sigma}^* \mid \boldsymbol{w}$ **is a Palindrome}**. Design a PDA P that recognizes L.

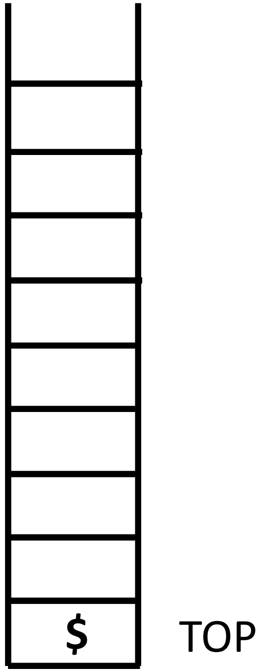
Intuition

- Push first half of the input string onto the stack.
- Verify that the second half of the symbols match the first half: Keep Popping the stack until the end of the input.

- How can the PDA know that the middle of the input has been reached.
    - The PDA does this non-deterministically (by taking ϵ transitions).







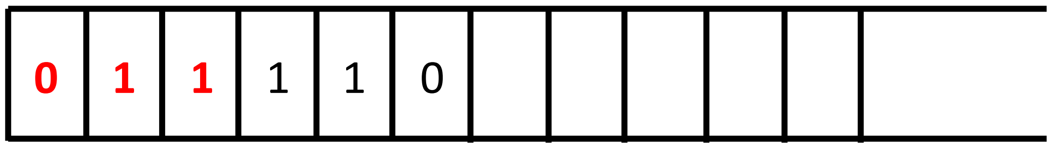
Pushdown Automata

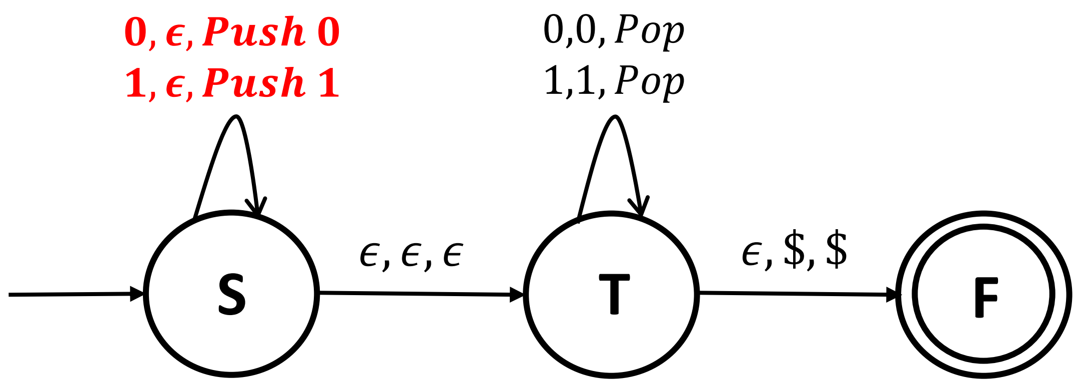
Let $\Sigma = \{0,1\}$ consider the language $\boldsymbol{L} = \{\boldsymbol{w} \in \boldsymbol{\Sigma}^* \mid \boldsymbol{w}$ **is a Palindrome**$\}$. Design a PDA P that recognizes L.

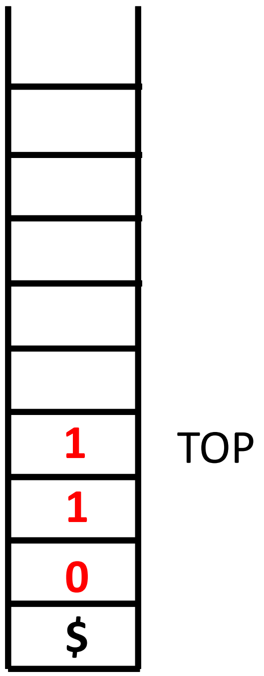
Intuition

- Push first half of the input string onto the stack.
- Verify that the second half of the symbols match the first half: Keep Popping the stack until the end of the input.

- How can the PDA know that the middle of the input has been reached.
  - The PDA does this non-deterministically (by taking ϵ transitions).







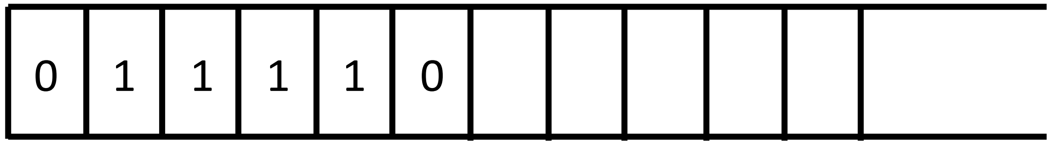
Pushdown Automata

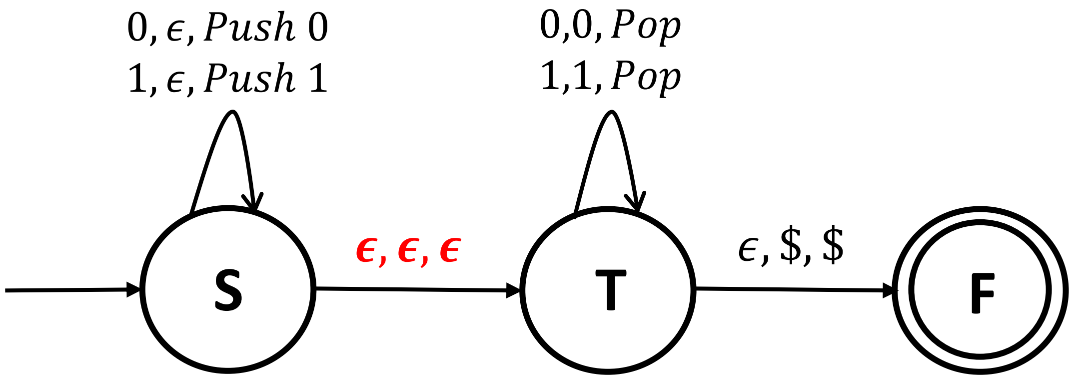
Let $\Sigma = \{0,1\}$ consider the language $\boldsymbol{L} = \{\boldsymbol{w} \in \boldsymbol{\Sigma}^* \mid \boldsymbol{w}$ **is a Palindrome}**. Design a PDA P that recognizes L.

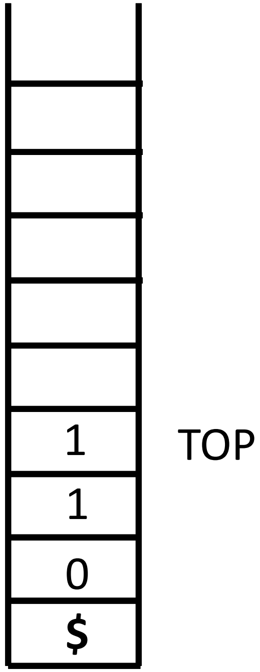
Intuition

- Push first half of the input string onto the stack.
- Verify that the second half of the symbols match the first half: Keep Popping the stack until the end of the input.

- How can the PDA know that the middle of the input has been reached.
  - The PDA does this non-deterministically (by taking ϵ transitions).







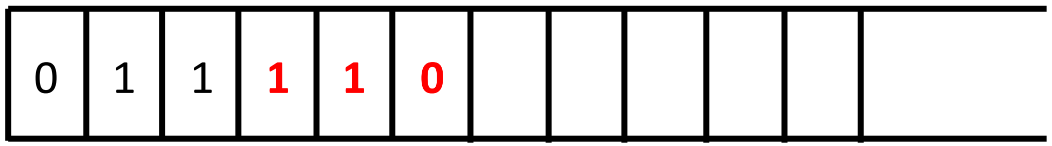
Pushdown Automata

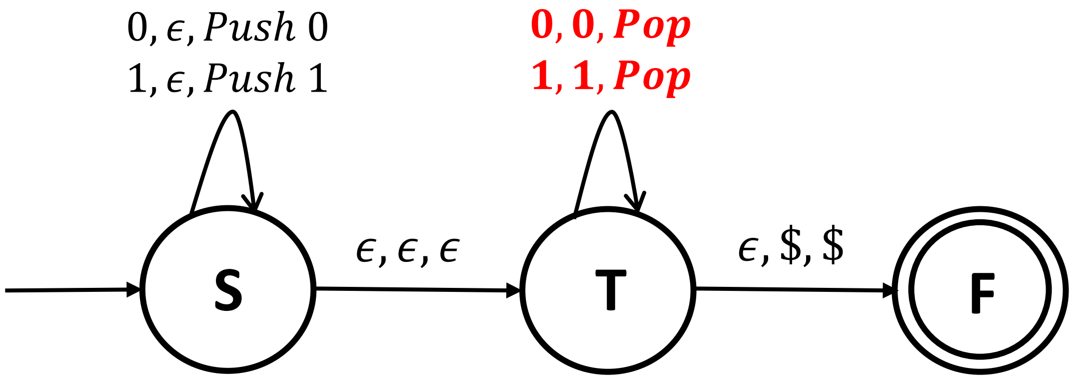
Let $\Sigma = \{0,1\}$ consider the language $\boldsymbol{L} = \{\boldsymbol{w} \in \boldsymbol{\Sigma}^* \mid \boldsymbol{w}$ **is a Palindrome**$\}$. Design a PDA P that recognizes L.

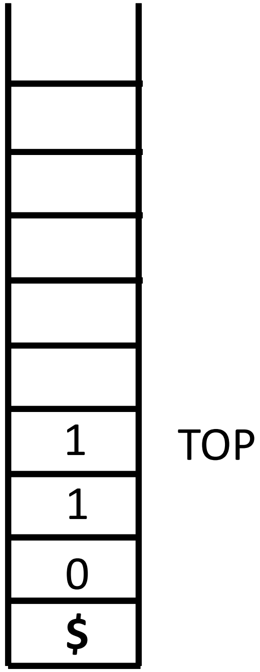
Intuition

- Push first half of the input string onto the stack.
- Verify that the second half of the symbols match the first half: Keep Popping the stack until the end of the input.

- How can the PDA know that the middle of the input has been reached.
  - The PDA does this non-deterministically (by taking ϵ transitions).







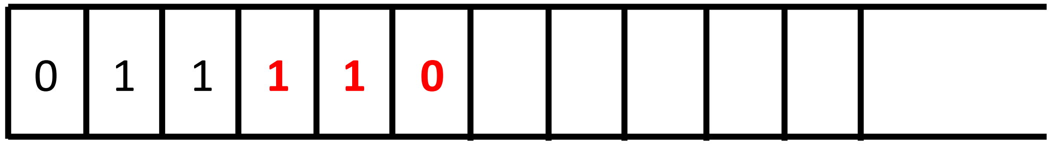
Pushdown Automata

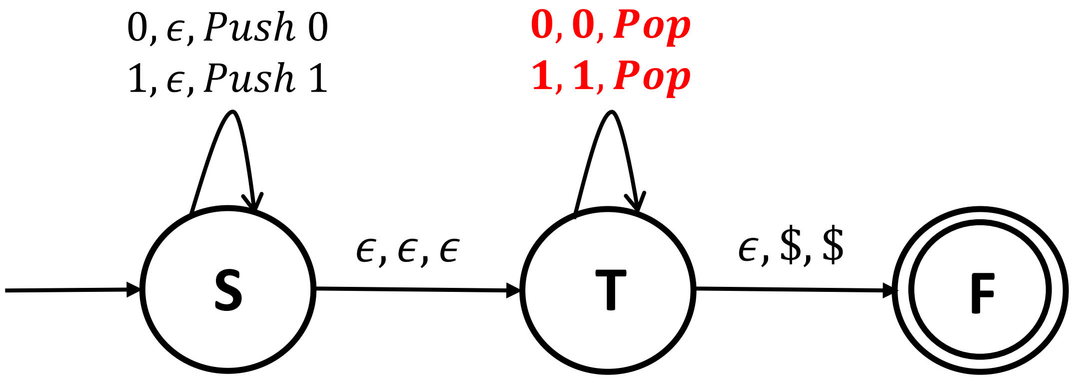
Let $\Sigma = \{0,1\}$ consider the language $\boldsymbol{L} = \{\boldsymbol{w} \in \boldsymbol{\Sigma}^* \mid \boldsymbol{w}$ **is a Palindrome**$\}$. Design a PDA P that recognizes L.

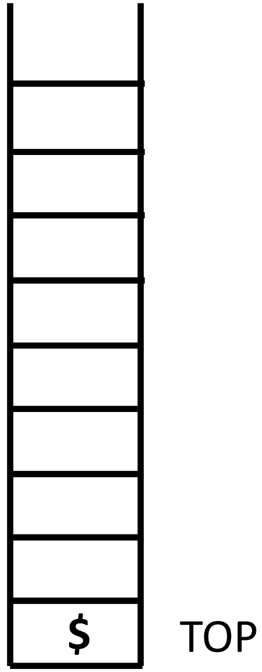
Intuition

- Push first half of the input string onto the stack.
- Verify that the second half of the symbols match the first half: Keep Popping the stack until the end of the input.

- How can the PDA know that the middle of the input has been reached.
  - The PDA does this non-deterministically (by taking ϵ transitions).







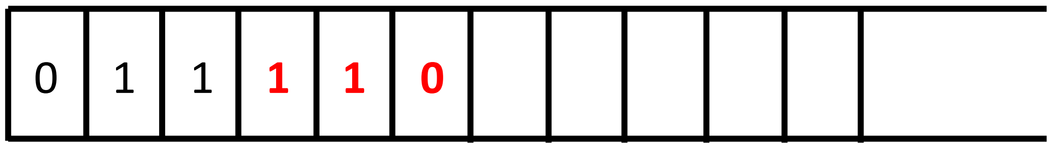
Pushdown Automata

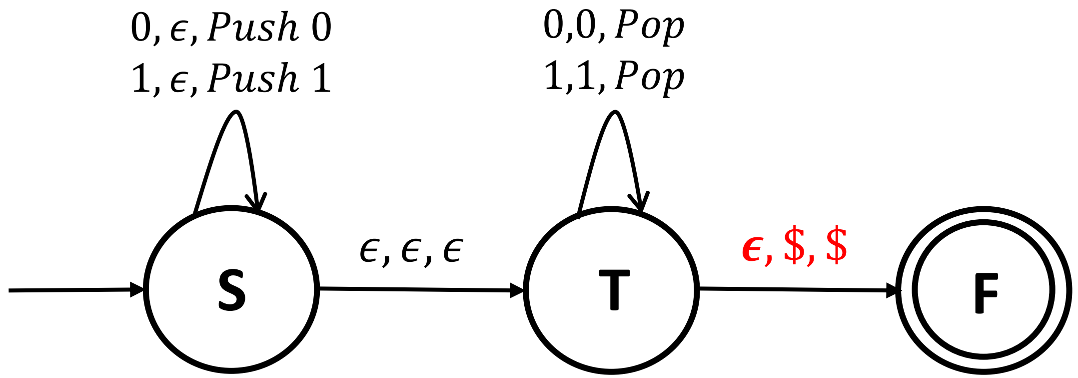
Let $\Sigma = \{0,1\}$ consider the language $\boldsymbol{L} = \{\boldsymbol{w} \in \boldsymbol{\Sigma}^* \mid$ **$\boldsymbol{w}$ is a Palindrome}**. Design a PDA P that recognizes L.

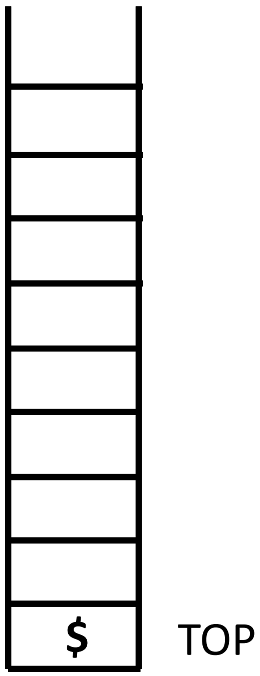
Intuition

- Push first half of the input string onto the stack.
- Verify that the second half of the symbols match the first half: Keep Popping the stack until the end of the input.

- How can the PDA know that the middle of the input has been reached.
  - The PDA does this non-deterministically (by taking ϵ transitions).







Pushdown Automata

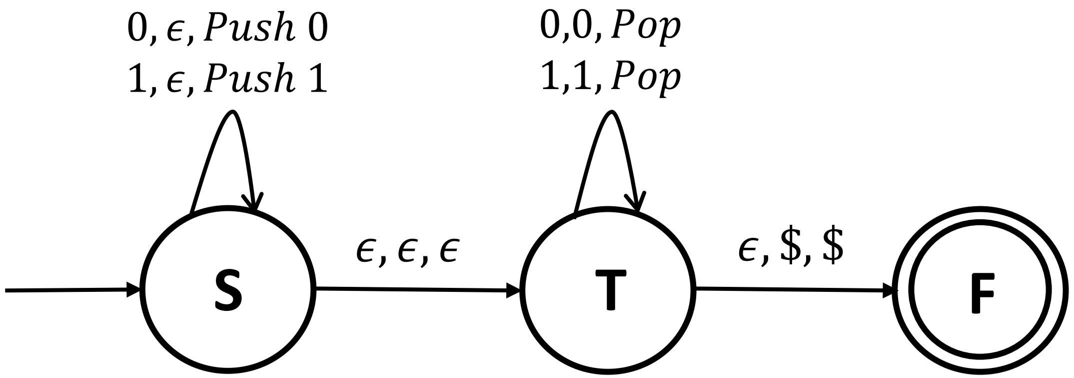
Let $\Sigma = \{0{,}1\}$ consider the language $\boldsymbol{L} = \{\boldsymbol{w} \in \boldsymbol{\Sigma}^* \mid \boldsymbol{w}$ **is a Palindrome}**. Design a PDA P that recognizes L.

Intuition

- Push first half of the input string onto the stack.
- Verify that the second half of the symbols match the first half: Keep Popping the stack until the end of the input.

- How can the PDA know that the middle of the input has been reached.
  - The PDA does this non-deterministically (by taking ϵ transitions).

- What about odd length palindromes?



Recognizes even length palindromes of the form: $\boldsymbol{ww^R}$

Pushdown Automata

Let $\Sigma = \{0,1\}$ consider the language $\boldsymbol{L} = \{\boldsymbol{w} \in \boldsymbol{\Sigma}^* \mid \boldsymbol{w}$ **is a Palindrome}**. Design a PDA P that recognizes L.

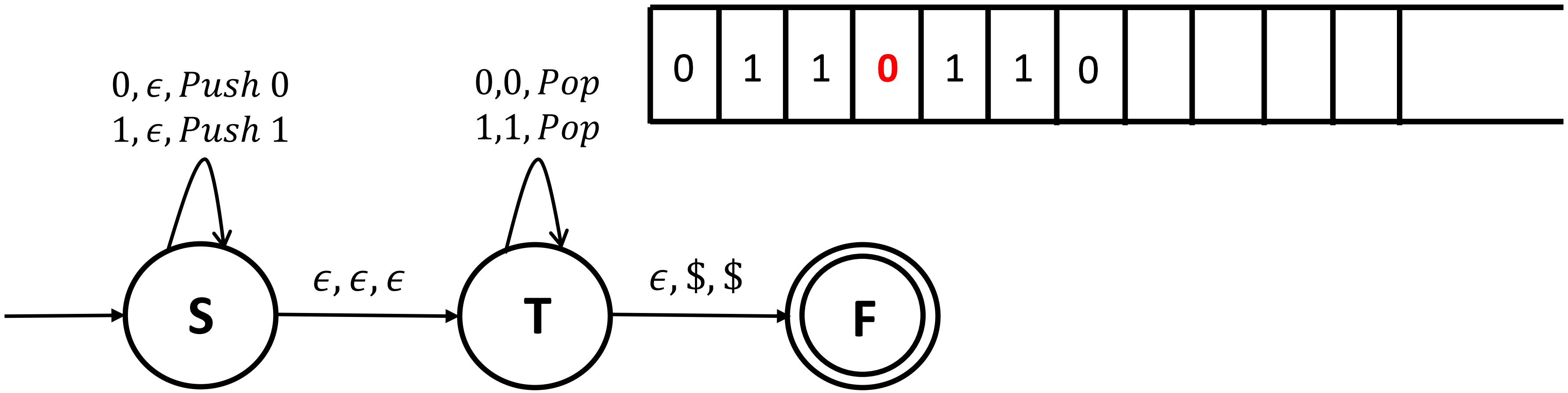
Intuition

- Push first half of the input string onto the stack.
- Verify that the second half of the symbols match the first half: Keep Popping the stack until the end of the input.

- How can the PDA know that the middle of the input has been reached.
  - The PDA does this non-deterministically (by taking ϵ transitions).

- What about odd length palindromes?

Odd length palindromes are of the form $\boldsymbol{wcw^R}$, such that $\boldsymbol{c \in \Sigma}$



Pushdown Automata

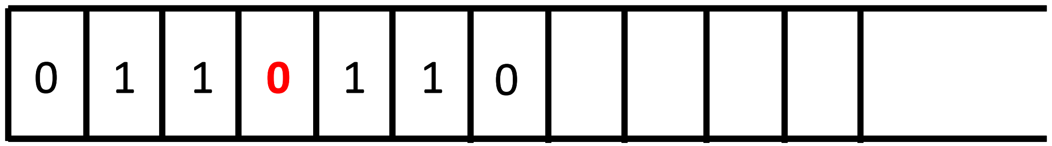
Let $\Sigma = \{0,1\}$ consider the language $\boldsymbol{L} = \{\boldsymbol{w} \in \boldsymbol{\Sigma}^* \mid \boldsymbol{w}$ **is a Palindrome}**. Design a PDA P that recognizes L.

Intuition

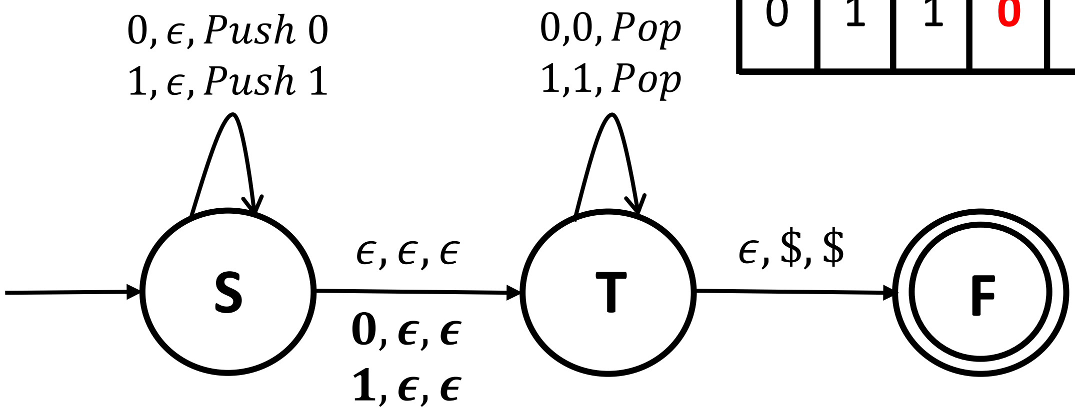
- Push first half of the input string onto the stack.
- Verify that the second half of the symbols match the first half: Keep Popping the stack until the end of the input.

- How can the PDA know that the middle of the input has been reached.
  - The PDA does this non-deterministically (by taking ϵ transitions).

- What about odd length palindromes?



Odd length palindromes are of the form $\boldsymbol{wcw^R}$, such that $\boldsymbol{c \in \Sigma}$



Pushdown Automata

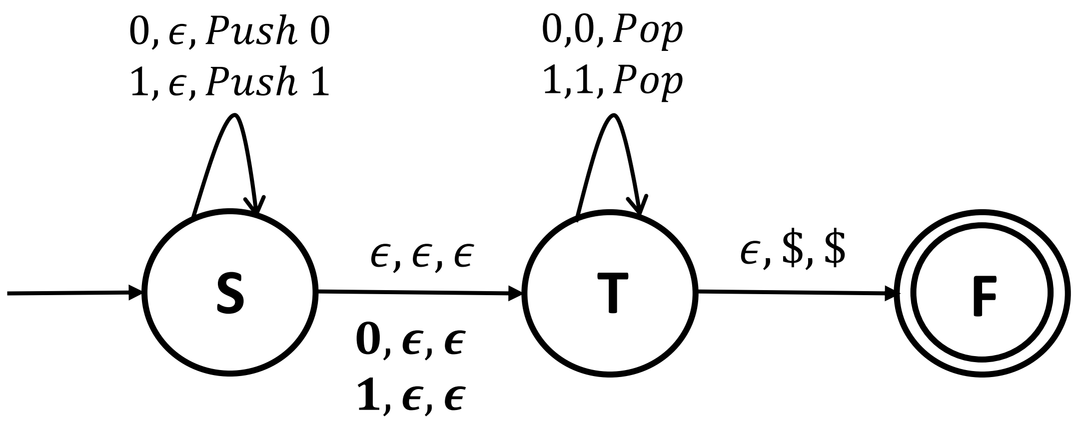
Let $\Sigma = \{0,1\}$ consider the language $\boldsymbol{L} = \{\boldsymbol{w} \in \boldsymbol{\Sigma}^* \mid \boldsymbol{w}$ **is a Palindrome}**. Design a PDA P that recognizes L.

Intuition

- Push first half of the input string onto the stack.
- Verify that the second half of the symbols match the first half: Keep Popping the stack until the end of the input.

- How can the PDA know that the middle of the input has been reached.
  - The PDA does this non-deterministically (by taking ϵ transitions).

- What about odd length palindromes?



The transitions $\boldsymbol{0, \epsilon, \epsilon}$ **and** $\boldsymbol{1, \epsilon, \epsilon}$ **allow the PDA to consume one symbol** and then begin matching what it has encountered thus far.

Pushdown Automata

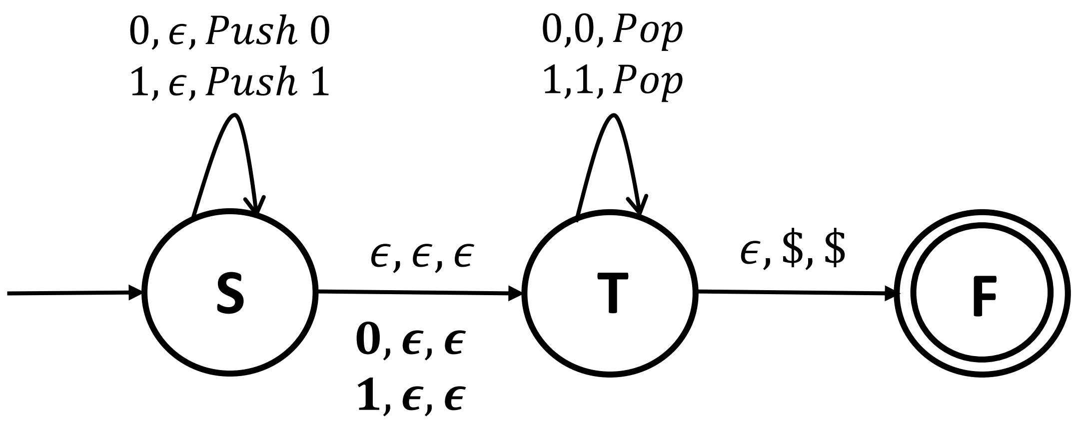
Let $\Sigma = \{0,1\}$ consider the language $\boldsymbol{L} = \{\boldsymbol{w} \in \boldsymbol{\Sigma}^* \mid \boldsymbol{w}$ **is a Palindrome}**. Design a PDA P that recognizes L.

Intuition

- Push first half of the input string onto the stack.
- Verify that the second half of the symbols match the first half: Keep Popping the stack until the end of the input.

- How can the PDA know that the middle of the input has been reached.
  - The PDA does this non-deterministically (by taking ϵ transitions).

- What about odd length palindromes?



The transitions $\mathbf{0}, \boldsymbol{\epsilon}, \boldsymbol{\epsilon}$ **and** $\mathbf{1}, \boldsymbol{\epsilon}, \boldsymbol{\epsilon}$ **allow the PDA to consume one symbol** and then begin matching what it has encountered thus far.

This allows the PDA to **recognize strings of the form:** $\boldsymbol{\omega c w^R}$, where the aforementioned transitions non-deterministically guessed $c \in \{0,1\}$

Thank You!