CS 302.1 - Automata Theory

Lecture 07

Shantanav Chakraborty

Center for Quantum Science and Technology (CQST)

Center for Security, Theory and Algorithms (CSTAR)

IIIT Hyderabad

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

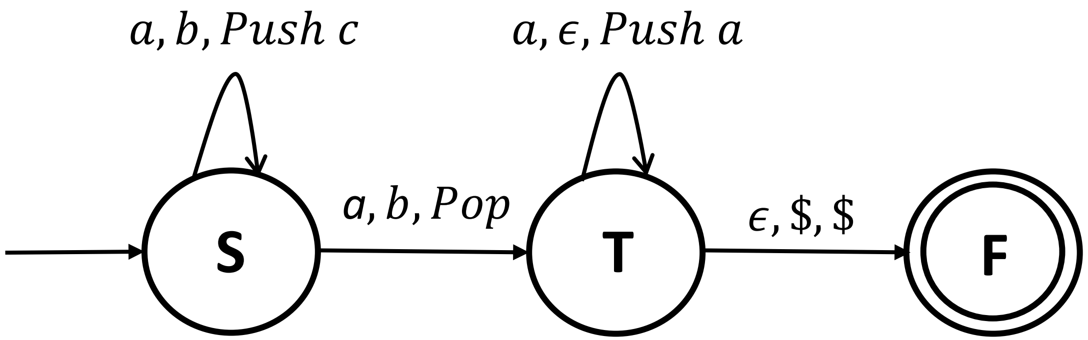
HYDERABAD

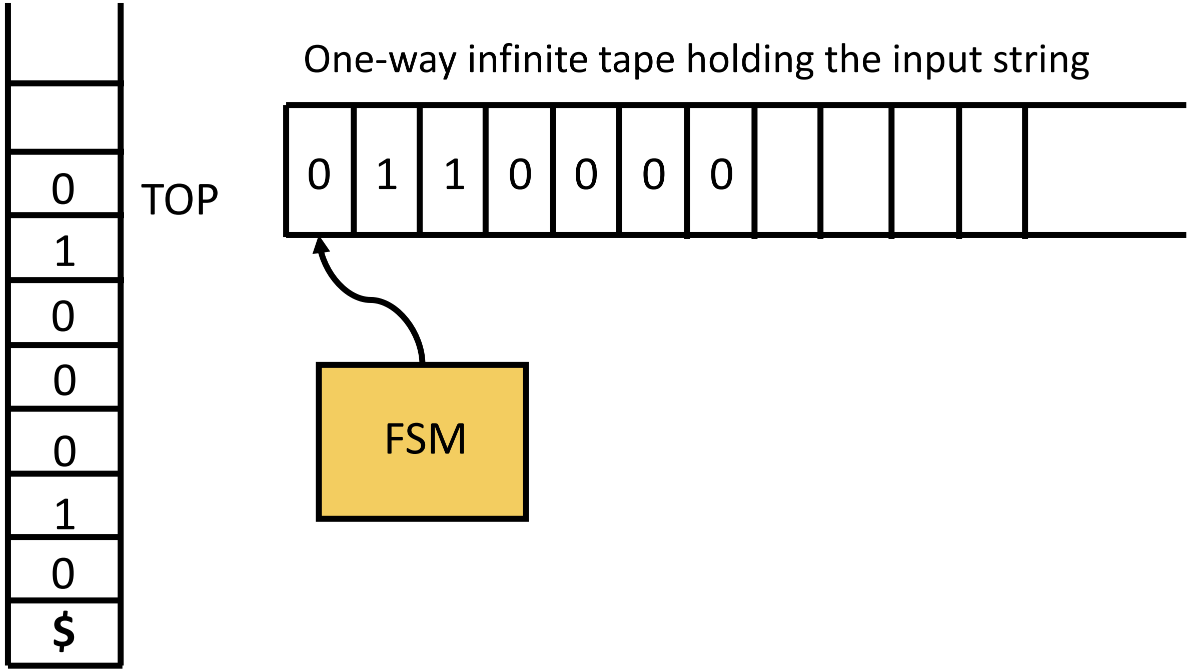
Quick Recap

Pushdown Automata

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

- $\boldsymbol{\delta(q_i, a, b) = (q_j, c)}$: If the input symbol read is a and the stack top = b, 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 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

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 $\boldsymbol{L} = \{\boldsymbol{w} \in \boldsymbol{\Sigma}^* \mid \boldsymbol{w}$ **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).

- 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

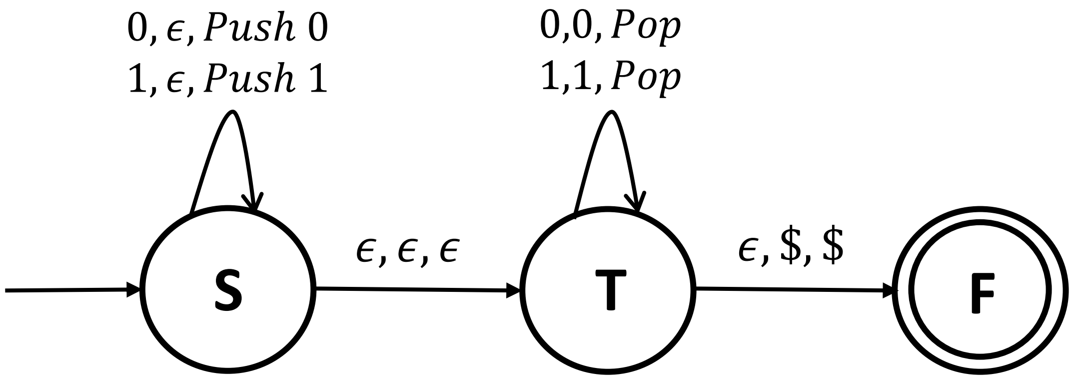
Pushdown Automata

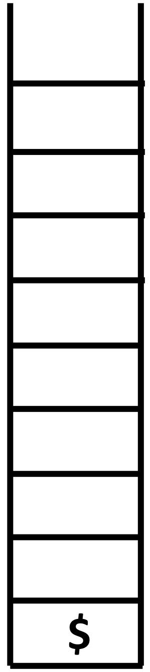
Let $\Sigma = \{0,1\}$ consider the language $\boldsymbol{L} = \{\boldsymbol{w} \in \boldsymbol{\Sigma}^* \mid \boldsymbol{w}$ **is a Palindrome}**. Design a PDA P that recognizes L.

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).





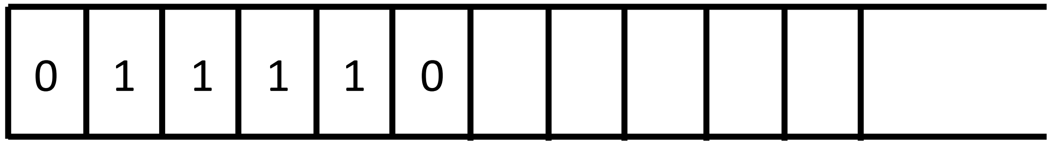
Pushdown Automata

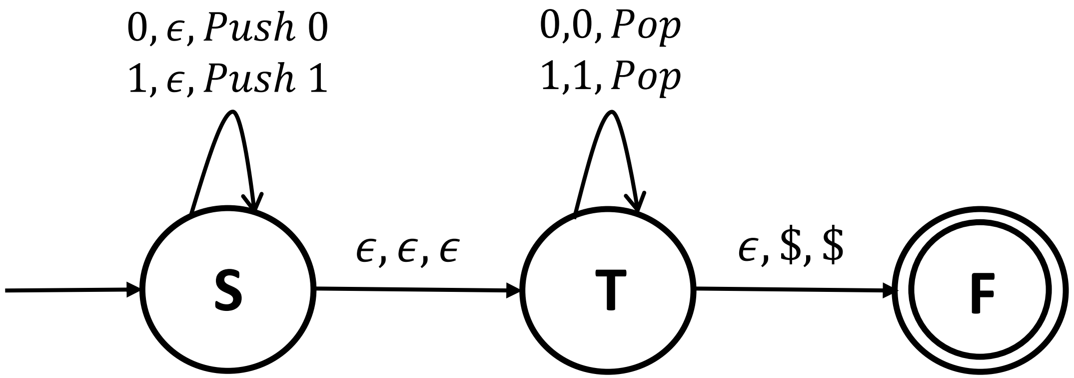
Let $\Sigma = \{0,1\}$ consider the language $\boldsymbol{L} = \{\boldsymbol{w} \in \boldsymbol{\Sigma}^* \ |\boldsymbol{w}$ **is a Palindrome}**. Design a PDA P that recognizes L.

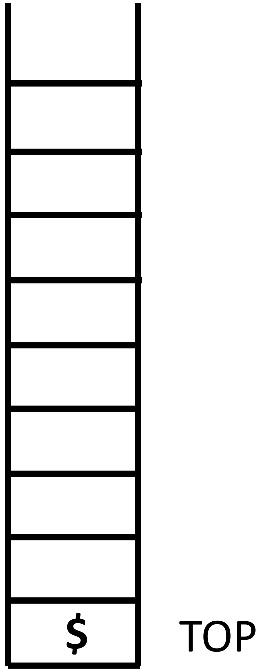
Intuition

- 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).







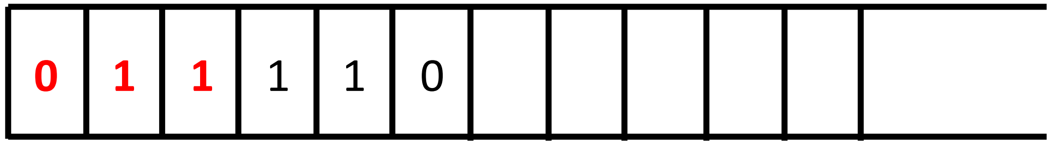
Pushdown Automata

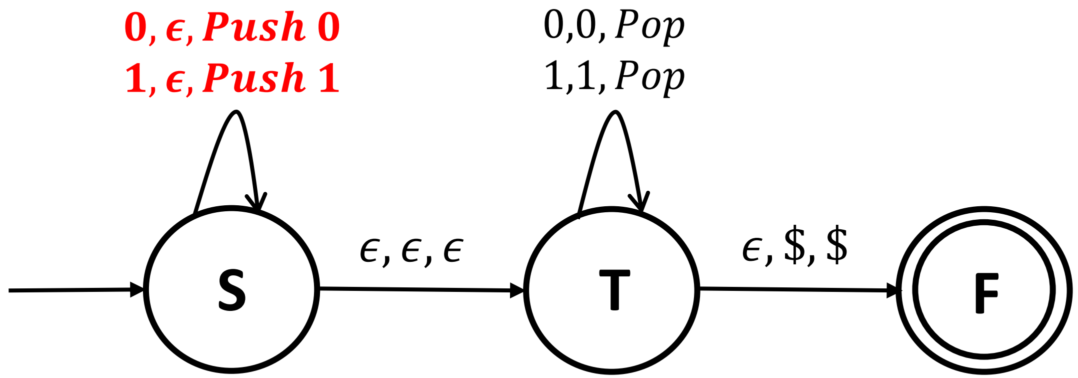
Let $\Sigma = \{0,1\}$ consider the language $\boldsymbol{L} = \{\boldsymbol{w} \in \boldsymbol{\Sigma}^* \; | \boldsymbol{w}$ **is a Palindrome}**. Design a PDA P that recognizes L.

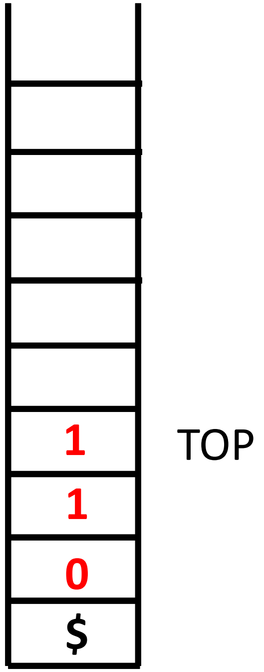
Intuition

- 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).







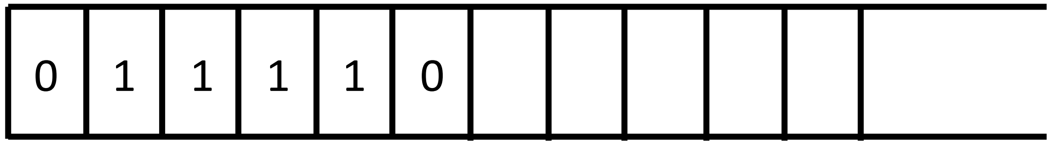
Pushdown Automata

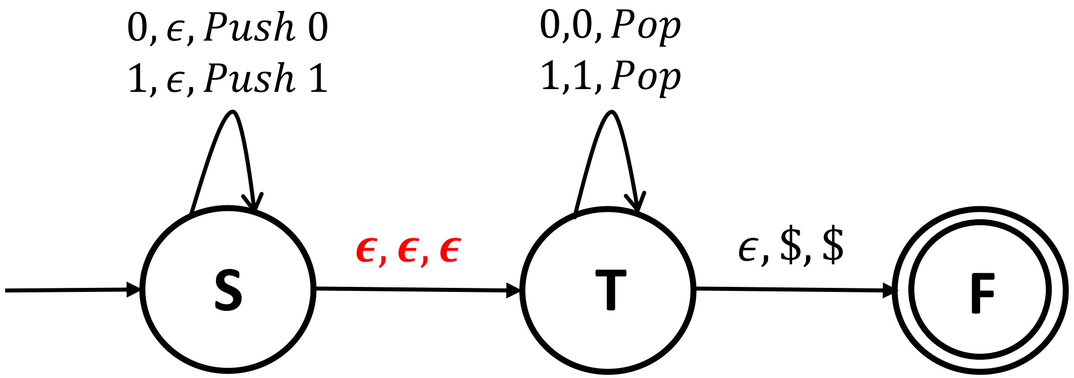
Let $\Sigma = \{0,1\}$ consider the language $\boldsymbol{L} = \{\boldsymbol{w} \in \boldsymbol{\Sigma}^* \mid \boldsymbol{w}$ **is a Palindrome**$\}$. Design a PDA P that recognizes L.

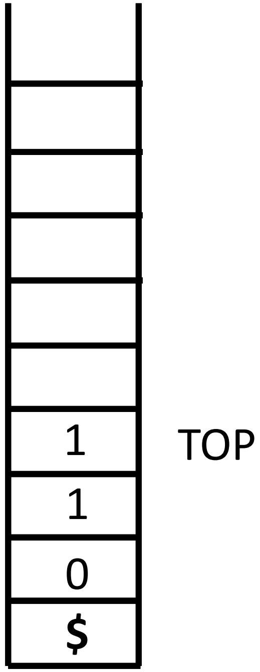
Intuition

- 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).







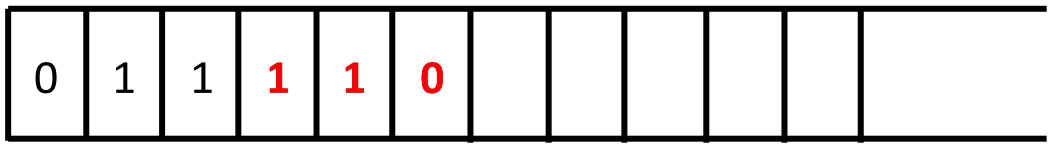
Pushdown Automata

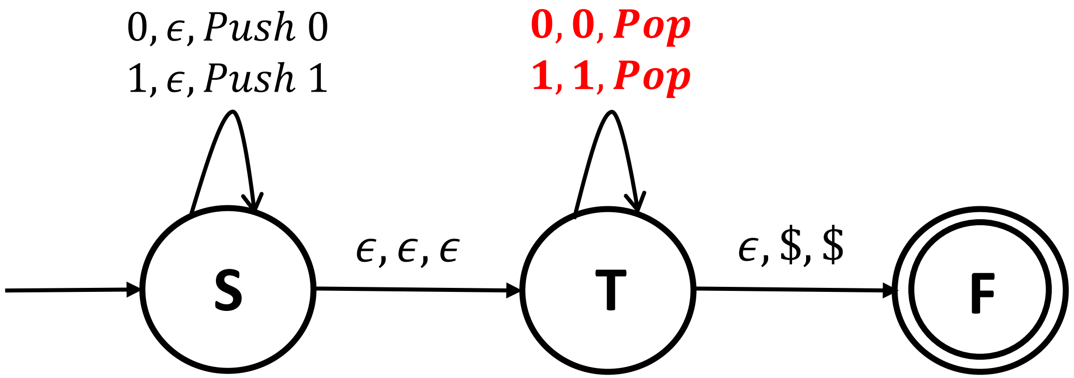
Let $\Sigma = \{0{,}1\}$ consider the language $\boldsymbol{L} = \{\boldsymbol{w} \in \boldsymbol{\Sigma}^* \,|\boldsymbol{w}$ **is a Palindrome}**. Design a PDA P that recognizes L.

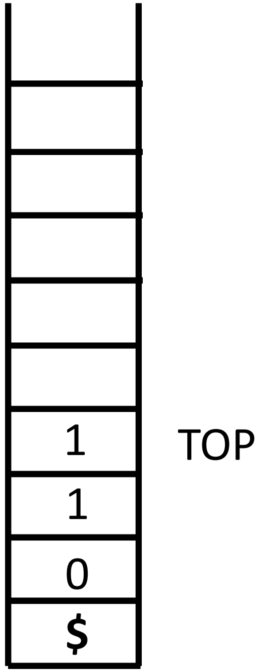
Intuition

- 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).







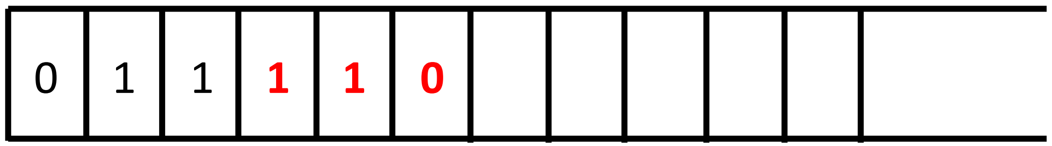
Pushdown Automata

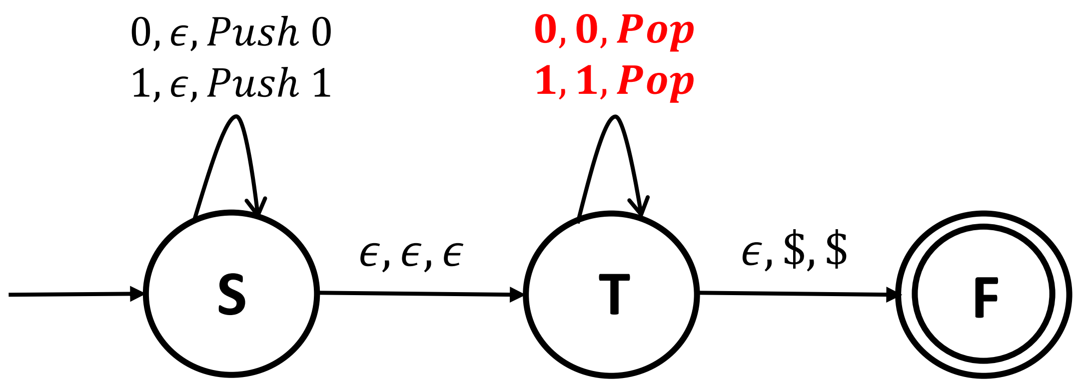
Let $\Sigma = \{0,1\}$ consider the language $\boldsymbol{L} = \{\boldsymbol{w} \in \boldsymbol{\Sigma}^* \mid \boldsymbol{w}$ **is a Palindrome}**. Design a PDA P that recognizes L.

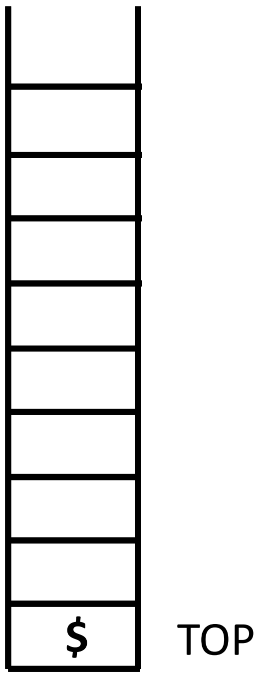
Intuition

- 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).







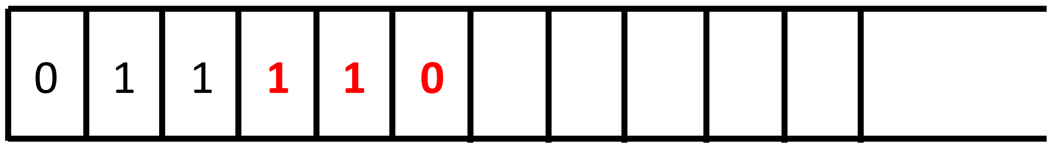
Pushdown Automata

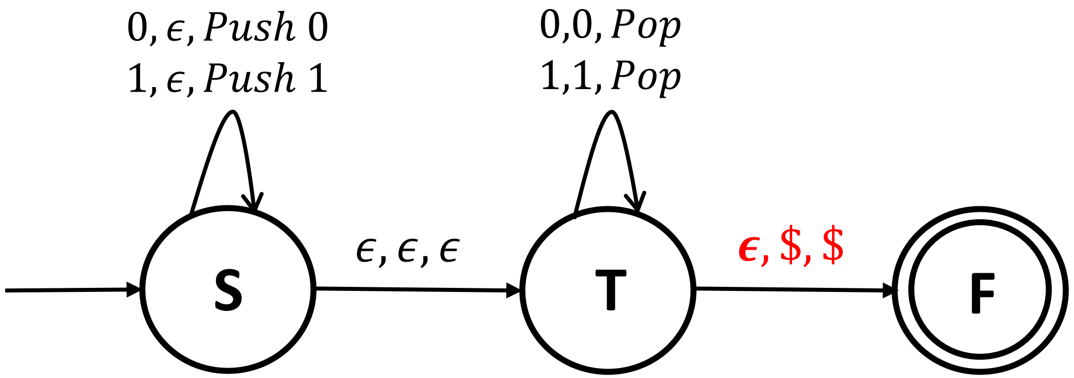
Let $\Sigma = \{0,1\}$ consider the language $\boldsymbol{L} = \{\boldsymbol{w} \in \boldsymbol{\Sigma}^* \mid \boldsymbol{w}$ **is a Palindrome}**. Design a PDA P that recognizes L.

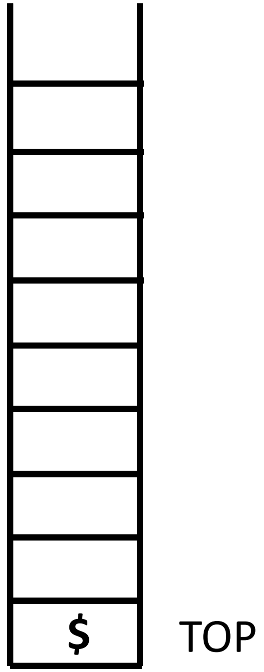
Intuition

- 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

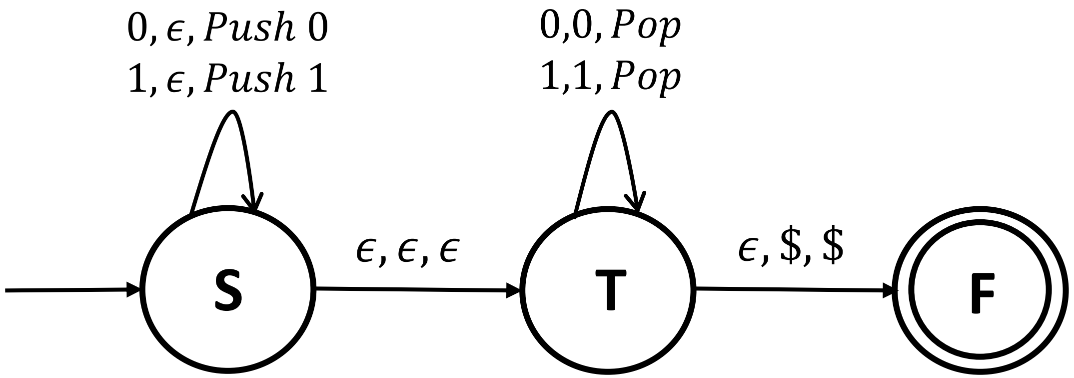
Let $\Sigma = \{0,1\}$ consider the language $\boldsymbol{L} = \{\boldsymbol{w} \in \boldsymbol{\Sigma}^* \mid$ **w is a Palindrome}**. Design a PDA P that recognizes L.

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: ww^R

Pushdown Automata

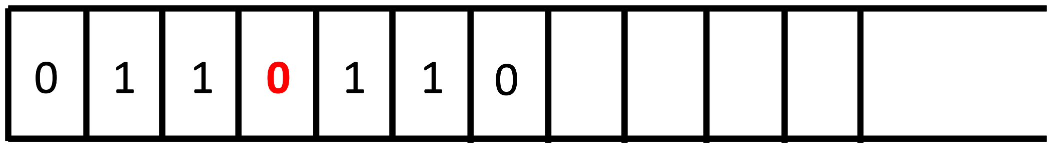
Let $\Sigma = \{0,1\}$ consider the language $\boldsymbol{L} = \{\boldsymbol{w} \in \boldsymbol{\Sigma}^* \ |\boldsymbol{w}$ **is a Palindrome}**. Design a PDA P that recognizes L.

Intuition

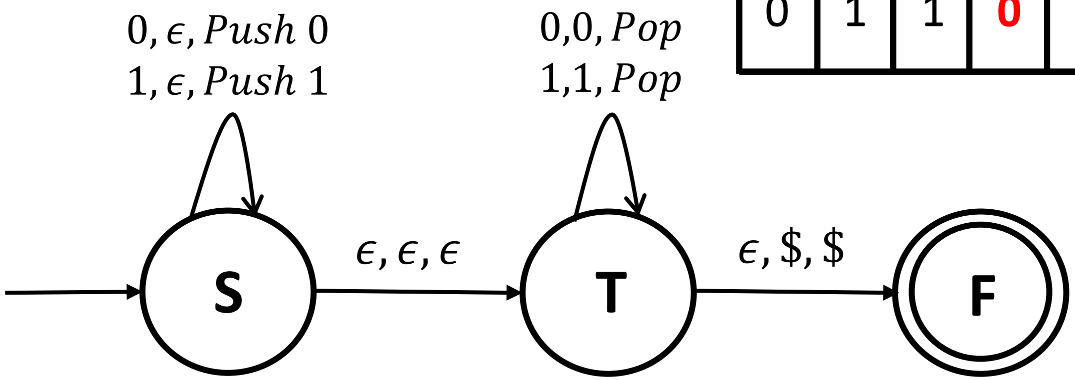
- Push first half of the input string onto the stack.
- Verify that the second half of the symbols match the first half: Keep Popping the stack until the end of the input.

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

- What about odd length palindromes?



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



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.

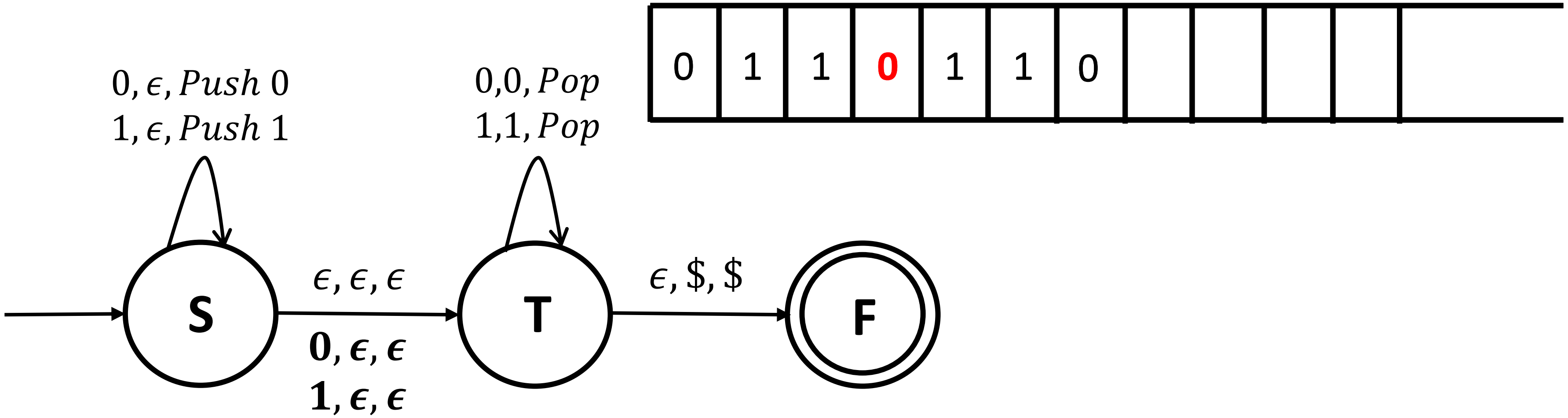
Intuition

- 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

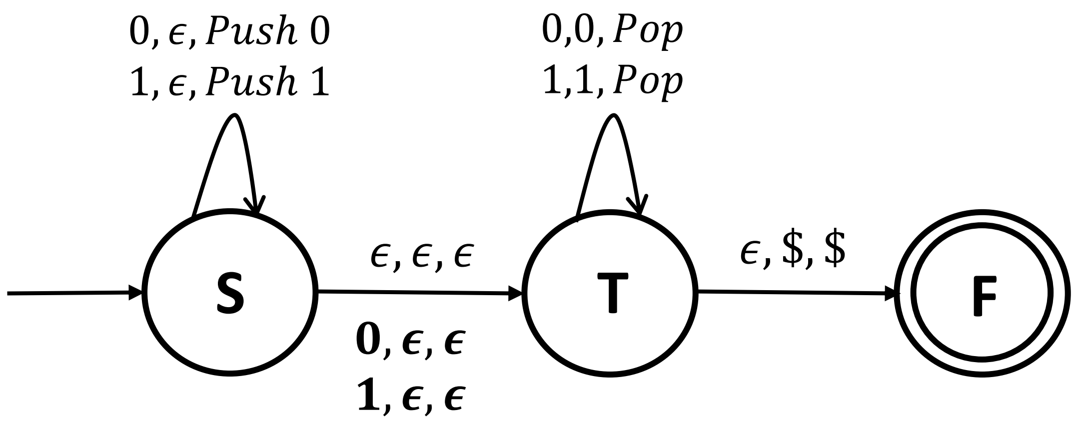
Let $\Sigma = \{0,1\}$ consider the language $\boldsymbol{L} = \{\boldsymbol{w} \in \boldsymbol{\Sigma}^* \mid \boldsymbol{w}$ **is a Palindrome}**. Design a PDA P that recognizes L.

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

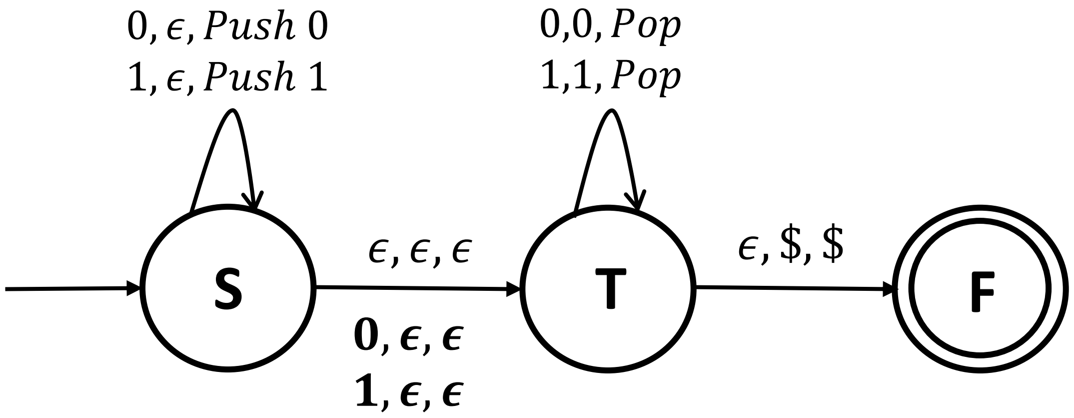
Let $\Sigma = \{0,1\}$ consider the language $\boldsymbol{L} = \{\boldsymbol{w} \in \boldsymbol{\Sigma}^* \mid \boldsymbol{w}$ **is a Palindrome}**. Design a PDA P that recognizes L.

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 $\boldsymbol{0}, \boldsymbol{\epsilon}, \boldsymbol{\epsilon}$ **and** $\boldsymbol{1}, \boldsymbol{\epsilon}, \boldsymbol{\epsilon}$ **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 $c \in \{0,1\}$

Equivalence between PDA and CFL

- We already know that a language is Context-Free if and only if there exists a CFG that generates all the strings belonging to the CFL.

- It can be shown that a language is context free if and only if a PDA recognizes it.

    - If L is context free then there exists a PDA that recognizes L. (We'll prove this next)

    - If there exists a PDA for L, then L is context-free. (Won't prove this in class. Look up a standard text book)

Pushdown Automata and CFL

Prove that if L is context free then there exists an equivalent PDA that recognizes L.

- Before formally proving this, we will use some examples in order to provide some intuition.
- For any L, we can write a context free grammar that can generate all strings that are in L.
- Any string w is generated by the CFG if there exists a derivation $S \overset{*}{\Rightarrow} w$.

Pushdown Automata and CFL

Prove that if L is context free then there exists an equivalent PDA that recognizes L.

- Before formally proving this, we will use some examples in order to provide some intuition.
- For any L, we can write a context free grammar that can generate all strings that are in L.
- Any string w is generated by the CFG if there exists a derivation $S \overset{*}{\Rightarrow} w$.
- The proof consists of using the rules of the CFG to build a PDA so that it can simulate any derivation $S \overset{*}{\Rightarrow} w$.
  - The PDA accepts an input w if the CFG G generates w
  - It determines whether $\exists$ a derivation for w.
  - Takes advantage of non determinism

Pushdown Automata and CFL

Prove that if L is context free then there exists an equivalent PDA that recognizes L.

Intuitions

- The PDA begins by pushing the start variable S onto the stack.

Pushdown Automata and CFL

Prove that if L is context free then there exists an equivalent PDA that recognizes L.

Intuitions

- The PDA begins by pushing the start variable S onto the stack.
- If the top of the stack is any variable A, then non-deterministically select one of the rules $A \to x$ (x can be a sequence of variables and terminals) pop A and push x on to the stack. **[Non deterministically chooses a rule as an intermediate derivation step]**

Pushdown Automata and CFL

Prove that if L is context free then there exists an equivalent PDA that recognizes L.

Intuitions

- The PDA begins by pushing the start variable S onto the stack.
- If the top of the stack is any variable A, then non-deterministically select one of the rules $A \rightarrow x$ (x can be a sequence of variables and terminals) pop A and push x on to the stack. **[Non deterministically chooses a rule as an intermediate derivation step]**
- **Read the input symbol** if the top of the stack is some terminal a. **[This tries to match part of the input string w non-deterministically]**

Pushdown Automata and CFL

Prove that if L is context free then there exists an equivalent PDA that recognizes L.

- The PDA begins by pushing the start variable S onto the stack.
- If the top of the stack is any variable A, then non-deterministically select one of the rules $A \to x$ (x can be a sequence of variables and terminals) pop A and push x on to the stack. **[Non deterministically chooses a rule as an intermediate derivation step]**
- **Read the input symbol** if the top of the stack is some terminal a. **[This tries to match part of the input string $\boldsymbol{w}$ non-deterministically]**

Example: Consider the grammar G with the rules: $S \to aTb|b$

$T \to Ta|\epsilon$

The string $\boldsymbol{w} = \boldsymbol{aaab}$ can be generated by G. Derivation:

$$\boldsymbol{S \to aTb \to aTab \to aTaab \to aaab}$$

Pushdown Automata and CFL

- The PDA begins by pushing the start variable S onto the stack.

- If the top of the stack any variable A, then non-deterministically select one of the rules $A \rightarrow x$ (x can be a sequence of variables and terminals) pop A and push x on to the stack. **[Non deterministically chooses a rule as an intermediate derivation step]**

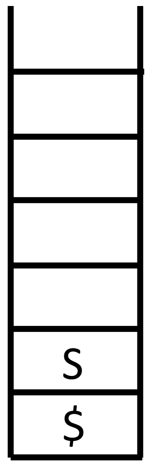
- **Read the input symbol** if the top of the stack is some terminal a. **[This tries to match part of the input string $\boldsymbol{w}$ non-deterministically]**

Example: $S \rightarrow aTb|b$
$T \rightarrow Ta|\epsilon$
Input to PDA: w $= \boldsymbol{aaab}$

Derivation for input string $w = aaab$ can be generated by G:

$$S \rightarrow aTb \rightarrow aTab \rightarrow aTaab \rightarrow aaab$$

1. Push S onto the Stack.



Pushdown Automata and CFL

- The PDA begins by pushing the start variable S onto the stack.

- If the top of the stack any variable A, then non-deterministically select one of the rules $A \rightarrow x$ (x can be a sequence of variables and terminals) pop A and push x on to the stack. **[Non deterministically chooses a rule as an intermediate derivation step]**

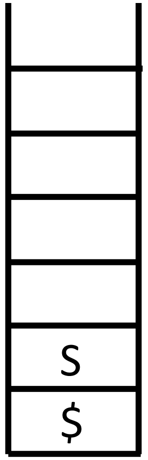
- **Read the input symbol** if the top of the stack is some terminal a. **[This tries to match part of the input string $\boldsymbol{w}$ non-deterministically]**

Example: $S \rightarrow aTb|b$
$T \rightarrow Ta|\epsilon$
Input to PDA: w $= \boldsymbol{aaab}$

Derivation for input string $w = aaab$ can be generated by G:

$$S \rightarrow aTb \rightarrow aTab \rightarrow aTaab \rightarrow aaab$$

1. Push S onto the Stack.
2. Pop S and
   a. Push b
   b. Push T
   c. Push a



Pushdown Automata and CFL

- The PDA begins by pushing the start variable S onto the stack.

- If the top of the stack any variable A, then non-deterministically select one of the rules $A \rightarrow x$ (x can be a sequence of variables and terminals) pop A and push x on to the stack. **[Non deterministically chooses a rule as an intermediate derivation step]**

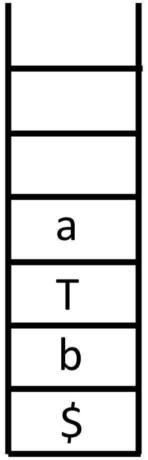
- **Read the input symbol** if the top of the stack is some terminal a. **[This tries to match part of the input string $\boldsymbol{w}$ non-deterministically]**

Example: $S \rightarrow aTb|b$
$T \rightarrow Ta|\epsilon$
Input to PDA: w $= \boldsymbol{aaab}$

Derivation for input string $w = aaab$ can be generated by G:

$$S \rightarrow aTb \rightarrow aTab \rightarrow aTaab \rightarrow aaab$$

1. Push S onto the Stack.
2. Pop S and
   a. Push b
   b. Push T
   c. Push a



Pushdown Automata and CFL

- The PDA begins by pushing the start variable S onto the stack.

- If the top of the stack any variable A, then non-deterministically select one of the rules $A \rightarrow x$ (x can be a sequence of variables and terminals) pop A and push x on to the stack. **[Non deterministically chooses a rule as an intermediate derivation step]**

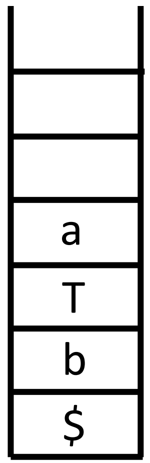
- **Read the input symbol** if the top of the stack is some terminal a. **[This tries to match part of the input string $\boldsymbol{w}$ non-deterministically]**

Example: $S \rightarrow aTb|b$
$T \rightarrow Ta|\epsilon$
Input to PDA: w $= \boldsymbol{aaab}$

Derivation for input string $w = \boldsymbol{a}aab$ can be generated by G:

$$S \rightarrow aTb \rightarrow aTab \rightarrow aTaab \rightarrow aaab$$

1. Push S onto the Stack.
2. Pop S and push aTb (Shorthand).
3. Read the input (a) and pop a.



Pushdown Automata and CFL

- The PDA begins by pushing the start variable S onto the stack.

- If the top of the stack any variable A, then non-deterministically select one of the rules $A \rightarrow x$ (x can be a sequence of variables and terminals) pop A and push x on to the stack. **[Non deterministically chooses a rule as an intermediate derivation step]**

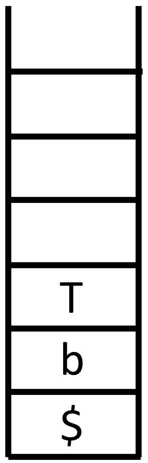
- **Read the input symbol** if the top of the stack is some terminal a. **[This tries to match part of the input string $\boldsymbol{w}$ non-deterministically]**

Example: $S \rightarrow aTb|b$
$T \rightarrow Ta|\epsilon$
Input to PDA: w $= \boldsymbol{aaab}$

Derivation for input string $w = \boldsymbol{a}aab$ can be generated by G:

$$S \rightarrow aTb \rightarrow aTab \rightarrow aTaab \rightarrow aaab$$

1. Push S onto the Stack.
2. Pop S and push aTb (Shorthand).
3. Read the input (a) and pop a.
4. Pop T and push Ta



Pushdown Automata and CFL

- The PDA begins by pushing the start variable S onto the stack.

- If the top of the stack any variable A, then non-deterministically select one of the rules $A \rightarrow x$ (x can be a sequence of variables and terminals) pop A and push x on to the stack. **[Non deterministically chooses a rule as an intermediate derivation step]**

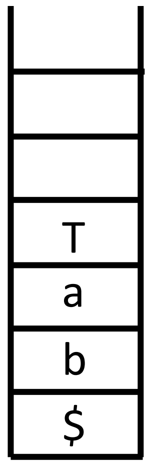
- **Read the input symbol** if the top of the stack is some terminal a. **[This tries to match part of the input string $\boldsymbol{w}$ non-deterministically]**

Example: $S \rightarrow aTb|b$
$T \rightarrow Ta|\epsilon$
Input to PDA: w $= \boldsymbol{aaab}$

Derivation for input string $w = \boldsymbol{a}aab$ can be generated by G:

$$S \rightarrow aTb \rightarrow aTab \rightarrow aTaab \rightarrow aaab$$

1. Push S onto the Stack.
2. Pop S and push aTb (Shorthand).
3. Read the input (a) and pop a.
4. Pop T and push Ta
5. Pop T and push Ta



Pushdown Automata and CFL

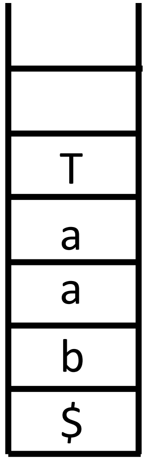
- The PDA begins by pushing the start variable S onto the stack.
- If the top of the stack any variable A, then non-deterministically select one of the rules $A \rightarrow x$ (x can be a sequence of variables and terminals) pop A and push x on to the stack. **[Non deterministically chooses a rule as an intermediate derivation step]**
- **Read the input symbol** if the top of the stack is some terminal a. **[This tries to match part of the input string $\boldsymbol{w}$ non-deterministically]**

Example: $S \rightarrow aTb|b$
$T \rightarrow Ta|\epsilon$
Input to PDA: w $= \boldsymbol{aaab}$

Derivation for input string $w = \boldsymbol{a}aab$ can be generated by G**:**

$$S \rightarrow aTb \rightarrow aTab \rightarrow aTaab \rightarrow aaab$$

1. Push S onto the Stack.
2. Pop S and push aTb (Shorthand).
3. Read the input (a) and pop a.
4. Pop T and push Ta
5. Pop T and push Ta
6. Pop T (for the rule $T \rightarrow \epsilon$)



Pushdown Automata and CFL

- The PDA begins by pushing the start variable S onto the stack.

- If the top of the stack any variable A, then non-deterministically select one of the rules $A \to x$ (x can be a sequence of variables and terminals) pop A and push x on to the stack. **[Non deterministically chooses a rule as an intermediate derivation step]**

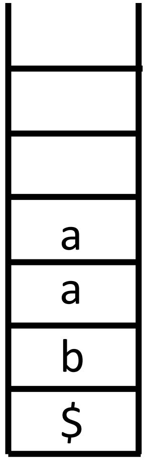
- **Read the input symbol** if the top of the stack is some terminal a. **[This tries to match part of the input string $\boldsymbol{w}$ non-deterministically]**

Example: $S \to aTb|b$
$T \to Ta|\epsilon$
Input to PDA: w $= \boldsymbol{aaab}$

Derivation for input string $w = \boldsymbol{aa}ab$ can be generated by G**:**

$$S \to aTb \to aTab \to aTaab \to aaab$$

1. Push S onto the Stack.
2. Pop S and push aTb (Shorthand).
3. Read the input (a) and pop a.
4. Pop T and push Ta
5. Pop T and push Ta
6. Pop T (for the rule $T \to \epsilon$)
7. Read the input (a) and pop a.



Pushdown Automata and CFL

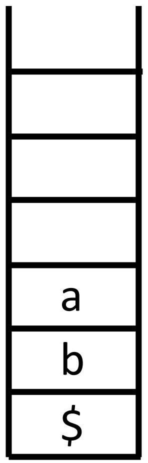
- The PDA begins by pushing the start variable S onto the stack.
- If the top of the stack any variable A, then non-deterministically select one of the rules $A \to x$ (x can be a sequence of variables and terminals) pop A and push x on to the stack. **[Non deterministically chooses a rule as an intermediate derivation step]**
- **Read the input symbol** if the top of the stack is some terminal a. **[This tries to match part of the input string w non-deterministically]**

Example: $S \to aTb|b$
$T \to Ta|\epsilon$
Input to PDA: w $= \boldsymbol{aaab}$

Derivation for input string $w = \boldsymbol{aaa}b$ can be generated by G**:**

$$S \to aTb \to aTab \to aTaab \to aaab$$

1. Push S onto the Stack.
2. Pop S and push aTb (Shorthand).
3. Read the input (a) and pop a.
4. Pop T and push Ta
5. Pop T and push Ta
6. Pop T (for the rule $T \to \epsilon$)
7. Read the input (a) and pop a.
8. Read the input (a) and pop a.



Pushdown Automata and CFL

- The PDA begins by pushing the start variable S onto the stack.

- If the top of the stack any variable A, then non-deterministically select one of the rules $A \rightarrow x$ (x can be a sequence of variables and terminals) pop A and push x on to the stack. **[Non deterministically chooses a rule as an intermediate derivation step]**

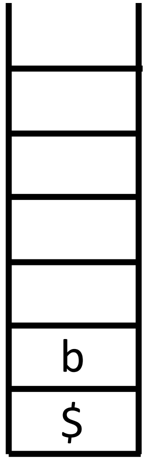
- **Read the input symbol** if the top of the stack is some terminal a. **[This tries to match part of the input string $\boldsymbol{w}$ non-deterministically]**

Example: $S \rightarrow aTb|b$
$T \rightarrow Ta|\epsilon$
Input to PDA: w $= \boldsymbol{aaab}$

Derivation for input string $w = \boldsymbol{aaab}$ can be generated by G:

$$S \rightarrow aTb \rightarrow aTab \rightarrow aTaab \rightarrow aaab$$

1. Push S onto the Stack.
2. Pop S and push aTb (Shorthand).
3. Read the input (a) and pop a.
4. Pop T and push Ta
5. Pop T and push Ta
6. Pop T (for the rule $T \rightarrow \epsilon$)
7. Read the input (a) and pop a.
8. Read the input (a) and pop a.
9. Read the input (b) and pop b.



Pushdown Automata and CFL

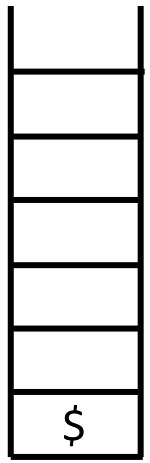
- The PDA begins by pushing the start variable S onto the stack.
- If the top of the stack any variable A, then non-deterministically select one of the rules $A \rightarrow x$ (x can be a sequence of variables and terminals) pop A and push x on to the stack. **[Non deterministically chooses a rule as an intermediate derivation step]**
- **Read the input symbol** if the top of the stack is some terminal a.

Example: $S \rightarrow aTb|b$
$T \rightarrow Ta|\epsilon$
Input to PDA: w $= \boldsymbol{aaab}$

Derivation for input string $w = \boldsymbol{aaab}$ can be generated by G:

$$S \rightarrow aTb \rightarrow aTab \rightarrow aTaab \rightarrow aaab$$

1. Push S onto the Stack.
2. Pop S and push aTb (Shorthand).
3. Read the input (a) and pop a.
4. Pop T and push Ta
5. Pop T and push Ta
6. Pop T (for the rule $T \rightarrow \epsilon$)
7. Read the input (a) and pop a.
8. Read the input (a) and pop a.
9. Read the input (b) and pop b.
10. Since the stack is empty exactly when the input has been read, accept w.



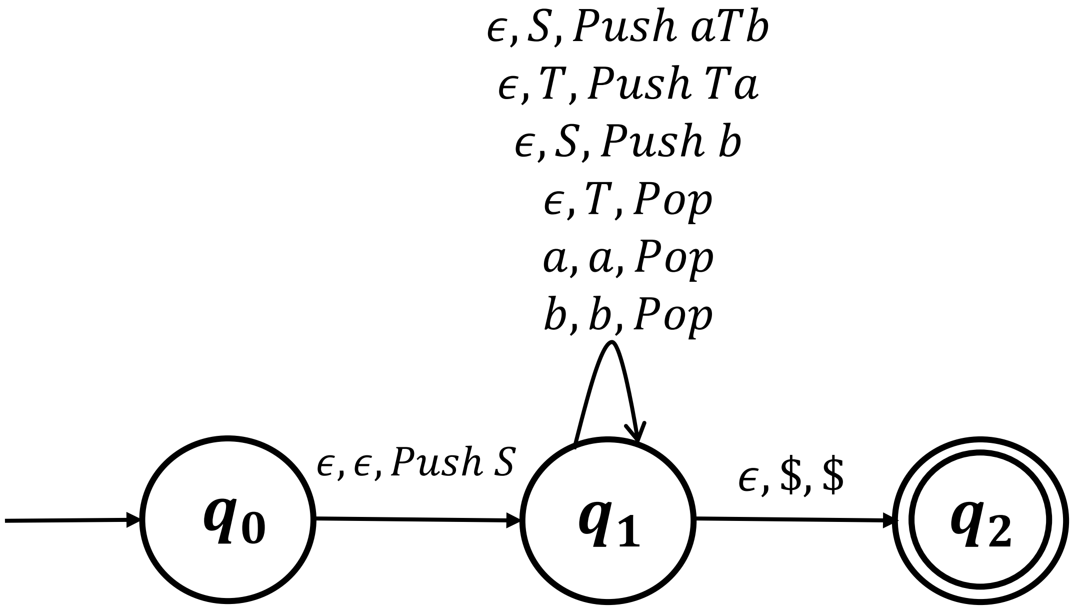
Pushdown Automata and CFL

Example: $S \rightarrow aTb|b$
$T \rightarrow Ta|\epsilon$

Input to PDA: $\mathbf{w} = \boldsymbol{aaab}$

Derivation for input string $w = \boldsymbol{aaab}$ can be generated by G:

$$S \rightarrow aTb \rightarrow aTab \rightarrow aTaab \rightarrow aaab$$



Pushdown Automata and CFL

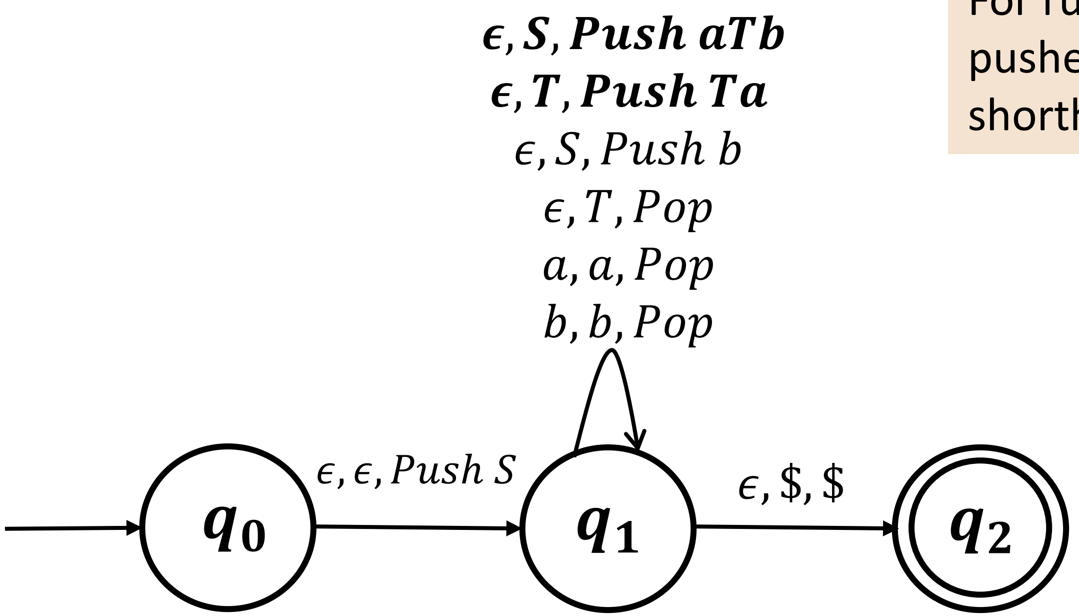
Example: $S \rightarrow aTb|b$
$T \rightarrow Ta|\epsilon$

Input to PDA: $\mathbf{w} = \boldsymbol{aaab}$

Derivation for input string $w = \boldsymbol{aaab}$ can be generated by G:

$$S \rightarrow aTb \rightarrow aTab \rightarrow aTaab \rightarrow aaab$$

For rules where several elements need to be pushed, new states are introduced. This is only a shorthand for that.



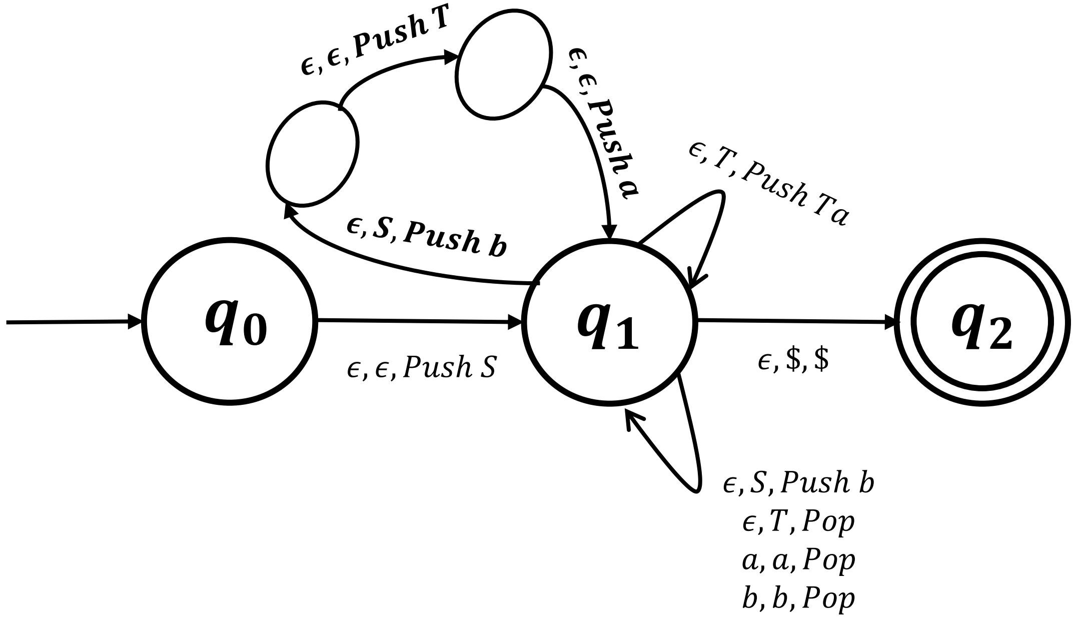
Pushdown Automata and CFL

Example: $S \rightarrow aTb|b$
$T \rightarrow Ta|\epsilon$

Input to PDA: $\mathbf{w} = \boldsymbol{aaab}$

Derivation for input string $w = \boldsymbol{aaab}$ can be generated by G:

$$S \rightarrow aTb \rightarrow aTab \rightarrow aTaab \rightarrow aaab$$



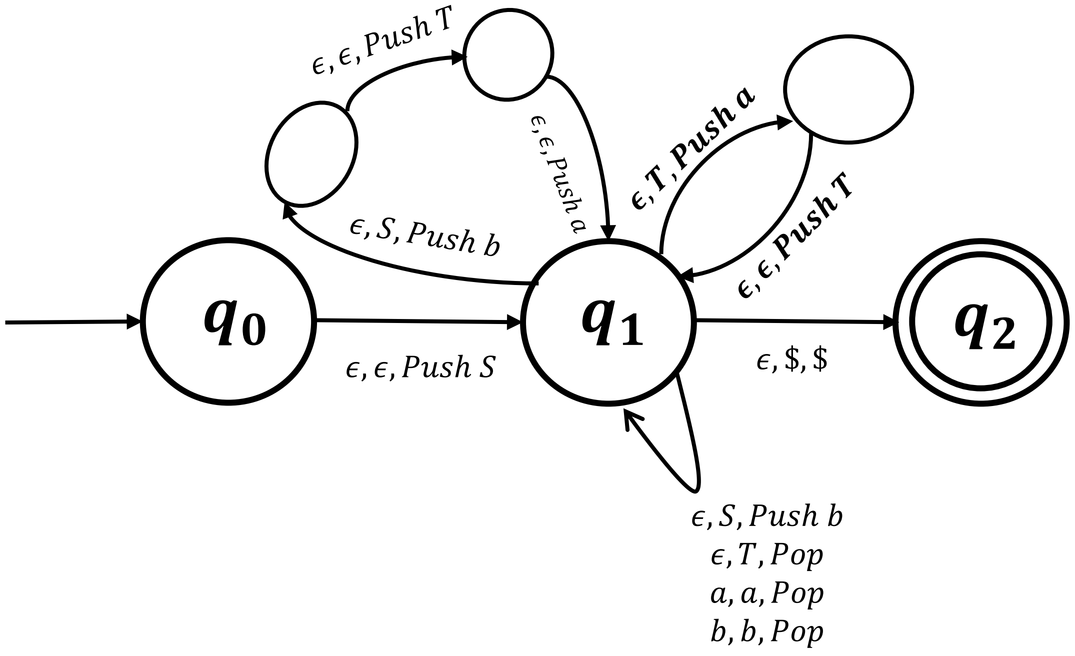
Pushdown Automata and CFL

Example: $S \rightarrow aTb|b$
$T \rightarrow Ta|\epsilon$

Input to PDA: $\mathbf{w} = \boldsymbol{aaab}$

Derivation for input string $w = \boldsymbol{aaab}$ can be generated by G:

$$S \rightarrow aTb \rightarrow aTab \rightarrow aTaab \rightarrow aaab$$



Summary

Given the rules of a CFG G, the equivalent PDA either non deterministically chooses which rule to use or matches part of the input symbol.

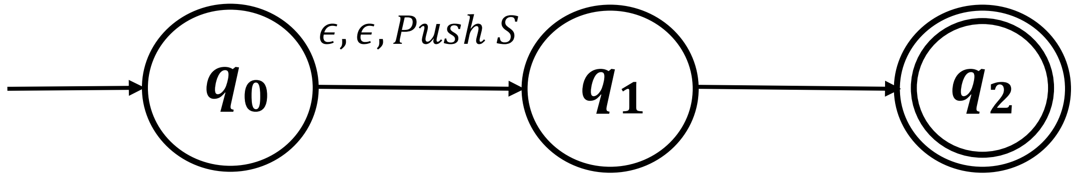
Pushdown Automata and CFL

Prove that if L is context free then there exists an equivalent PDA that recognizes L.

Proof: For convenience, we shall be using the shorthand notation.

Let G be a CFG with a set of rules R, then the equivalent PDA P will have three states $\{q_0, q_1, q_2\}$.

The PDA P first pushes the start symbol S into the stack, irrespective of the input symbol and transitions from the initial state q_0 to q_1, i.e. $\delta(q_0, \epsilon, \epsilon) = (q_1, S)$.



Pushdown Automata and CFL

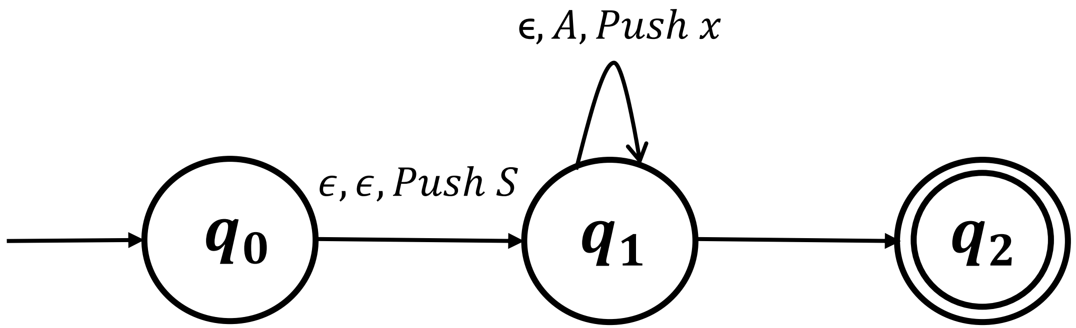
Prove that if L is context free then there exists an equivalent PDA that recognizes L.

Proof: Let G be a CFG with a set of rules R, then the equivalent PDA P will have three states $\{q_0, q_1, q_2\}$.

The PDA P first pushes the start symbol S into the stack, irrespective of the input symbol and transitions from the initial state q_0 to q_1, i.e. $\delta(q_0, \epsilon, \epsilon) = (q_1, S)$.

At q_1, the PDA P implements the rules R of G.

- Pop A and push x onto the stack, where $A \rightarrow x$ is a rule in R and return back to q_1, i.e. let $\delta(q_1, \epsilon, A) = (q_1, x)$.



For rule $A \rightarrow x$ in R

Pushdown Automata and CFL

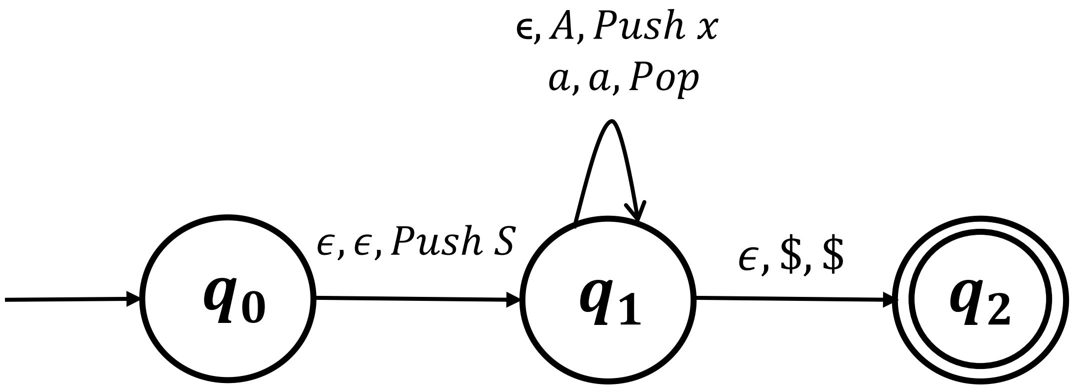
Prove that if L is context free then there exists an equivalent PDA that recognizes L.

Proof: Let G be a CFG with a set of rules R, then the equivalent PDA P will have three states $\{q_0, q_1, q_2\}$.

The PDA P first pushes the start symbol S into the stack, irrespective of the input symbol and transitions from the initial state q_0 to q_1, i.e. $\delta(q_0, \epsilon, \epsilon) = (q_1, S)$.

At q_1, the PDA P implements the rules R of G.

- Pop A and push x onto the stack, where $A \to x$ is a rule in R and return back to q_1, i.e. let $\delta(q_1, \epsilon, A) = (q_1, x)$.
- Read the input symbol a and pop a, i.e. let $\delta(q_1, a, a) = (q_1, \epsilon)$. Matching the input string with the terminals in stack.



For terminal a

Pushdown Automata and CFL

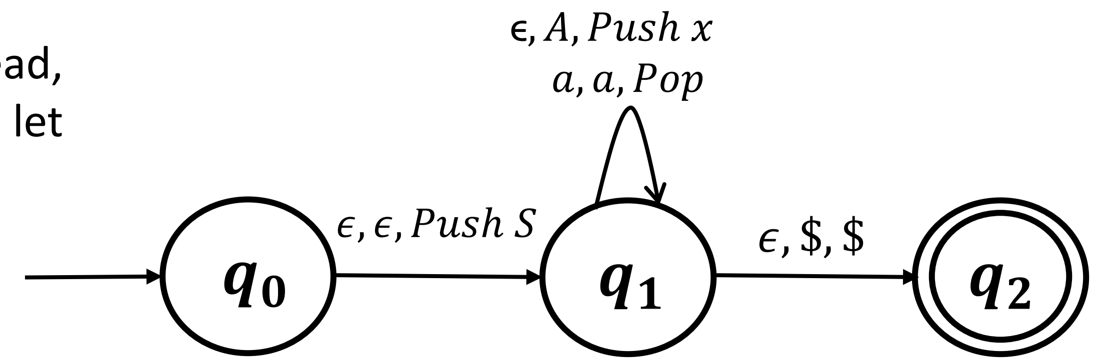
Prove that if L is context free then there exists an equivalent PDA that recognizes L.

Proof: Let G be a CFG with a set of rules R, then the equivalent PDA P will have three states $\{q_0, q_1, q_2\}$.

The PDA P first pushes the start symbol S into the stack, irrespective of the input symbol and transitions from the initial state q_0 to q_1, i.e. $\delta(q_0, \epsilon, \epsilon) = (q_1, S)$.

At q_1, the PDA P implements the rules R of G.

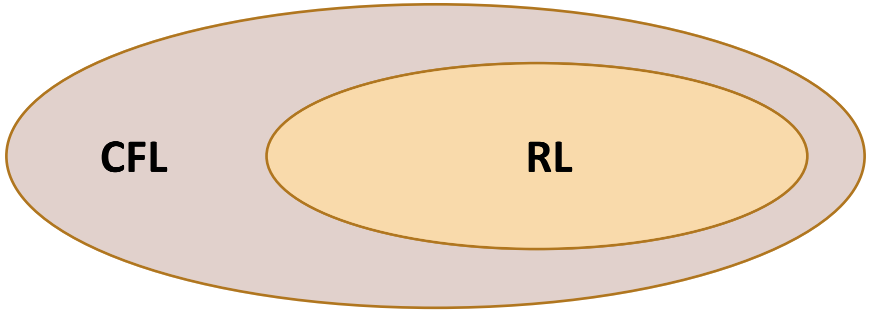
- Pop A and push x onto the stack, where $A \rightarrow x$ is a rule in R and return back to q_1, i.e. let $\delta(q_1, \epsilon, A) = (q_1, x)$.
- Read the input symbol a and pop a, i.e. let $\delta(q_1, a, a) = (q_1, \epsilon)$. Matching the input string with the terminals in stack.
- If the stack is empty, when all the input symbols are read, transition from q_1 to the accepting state q_2, i.e. let $\delta(q_1, \epsilon, \$) = (q_2, \$)$



Equivalence between PDA and CFL

- It can be shown that a language is context free **if and only if** a PDA recognizes it.
  - If L is context free then there exists a PDA that recognizes L. (We proved this)
  - The proof for the other direction (Constructing a CFG that generates L given a PDA that recognizes L) is quite elaborate
  - We won't be covering it in class. But the proof itself is quite easy to understand.
  - Refer to a standard text book (e.g. Sipser)

$$(\boldsymbol{RL \equiv Regular\ Grammar \equiv Regular\ Expressions \equiv NFA \equiv DFA}) \subseteq (\boldsymbol{CFL \equiv CFG \equiv PDA})$$



Deterministic Pushdown Automata

- So far we have considered Non-deterministic PDAs (which are referred to as just PDAs)
- Multiple transitions per input symbol/stack symbol is allowed
- Recall that for regular languages, introducing non-determinism added no extra power to finite automata: NFAs and DFAs were equivalent
- What about PDAs and CFLs?

Deterministic Pushdown Automata

- So far we have considered Non-deterministic PDAs (which are referred to as just PDAs)
- Multiple transitions per input symbol/stack symbol is allowed
- Recall that for regular languages, introducing non-determinism added no extra power to finite automata: NFAs and DFAs were equivalent
- What about PDAs and CFLs?

Deterministic Pushdown Automata (DPDA)

DPDAs can be defined in a similar manner to PDAs with the following restriction:

- For each state in the PDA, for any combination of the current input symbol and the current stack symbol **at most** one transition is possible. (At most one valid transition is allowed for any input)

Deterministic Pushdown Automata

- So far we have considered Non-deterministic PDAs (which are referred to as just PDAs)
- Multiple transitions per input symbol/stack symbol is allowed
- Recall that for regular languages, introducing non-determinism added no extra power to finite automata: NFAs and DFAs were equivalent
- What about PDAs and CFLs?

Deterministic Pushdown Automata (DPDA)

DPDAs can be defined in a similar manner to PDAs with the following restriction:

- For each state in the PDA, for any combination of the current input symbol and the current stack symbol **at most** one transition is possible. (At most one valid transition is allowed for any input)

- So determinism is defined slightly differently here. For example: ϵ transitions are allowed, some transitions may be absent (leading to CRASH runs) unlike in the case of DFAs.

Deterministic Pushdown Automata

DPDAs can be defined in a similar manner to PDAs with the following restriction:

- For each state in the PDA, for any combination of the current input symbol and the current stack symbol **at most** one transition is possible. (At most one valid transition is allowed for any input)

- So determinism is defined slightly differently here. For example: ϵ-transitions are allowed, some transitions may be absent (leading to CRASH runs) unlike in the case of DFAs.

- In fact, for a DPDA, the following condition must be satisfied:

  - For every $q \in Q, a \in \Sigma_\epsilon$ and $x \in \Gamma$, the set $\boldsymbol{\delta(q, a, x)}$ **has at most one element**

Deterministic Pushdown Automata

DPDAs can be defined in a similar manner to PDAs with the following restriction:

- For each state in the PDA, for any combination of the current input symbol and the current stack symbol **at most** one transition is possible. (At most one valid transition is allowed for any input)

- So determinism is defined slightly differently here. For example: ϵ-transitions are allowed, some transitions may be absent (leading to CRASH runs) unlike in the case of DFAs.

- In fact, for a DPDA, the following condition must be satisfied:

  - For every $q \in Q, a \in \Sigma_\epsilon$ and $x \in \Gamma$, the set $\boldsymbol{\delta(q, a, x)}$ **has at most one element**

This implies that if $\delta(q, \epsilon, x) \neq \Phi$, then $\delta(q, a, x) = \Phi$ for any $a \in \Sigma$: If there is an ϵ-transition for some configuration, no other input consuming move is possible.

Deterministic Pushdown Automata

DPDAs can be defined in a similar manner to PDAs with the following restriction:

- For each state in the PDA, for any combination of the current input symbol and the current stack symbol **at most** one transition is possible. (At most one valid transition is allowed for any input)

- So determinism is defined slightly differently here. For example: ϵ transitions are allowed, some transitions may be absent (leading to CRASH runs) unlike in the case of DFAs.

- In fact, for a DPDA, the transition rule δ must satisfy the following condition:

For every $q \in Q, a \in \Sigma$ and $x \in \Gamma$, **exactly one** of the following values is non-empty

$\boldsymbol{\delta(q, a, x), \delta(q, a, \epsilon), \delta(q, \epsilon, x)}$ **and** $\boldsymbol{\delta(q, \epsilon, \epsilon)}$

Deterministic Pushdown Automata

DPDAs can be defined in a similar manner to PDAs with the following restriction:

- For each state in the PDA, for any combination of the current input symbol and the current stack symbol **at most** one transition is possible. (At most one valid transition is allowed for any input)

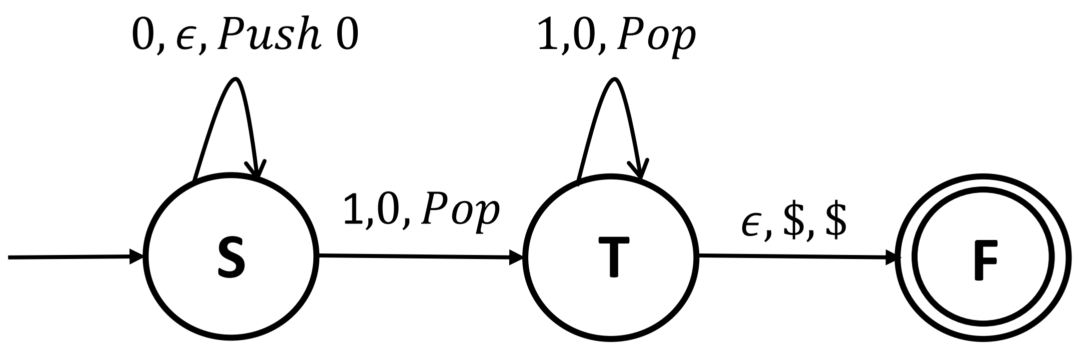
- So determinism is defined slightly differently here. For example: ϵ transitions are allowed, some transitions may be absent (leading to CRASH runs) unlike in the case of DFAs.

- In fact, for a DPDA, the transition rule δ must satisfy the following condition:

For every $q \in Q, \mathrm{a} \in \Sigma_\epsilon$ and $x \in \Gamma$, **exactly one** of the following values is non-empty

$$\boldsymbol{\delta(q,a,x), \delta(q,a,\epsilon), \delta(q,\epsilon,x)} \textbf{ and } \boldsymbol{\delta(q,\epsilon,\epsilon)}$$

Is this a DPDA? **YES!**



Deterministic Pushdown Automata

DPDAs can be defined in a similar manner to PDAs with the following restriction:

- For each state in the PDA, for any combination of the current input symbol and the current stack symbol **at most** one transition is possible. (At most one valid transition is allowed for any input)

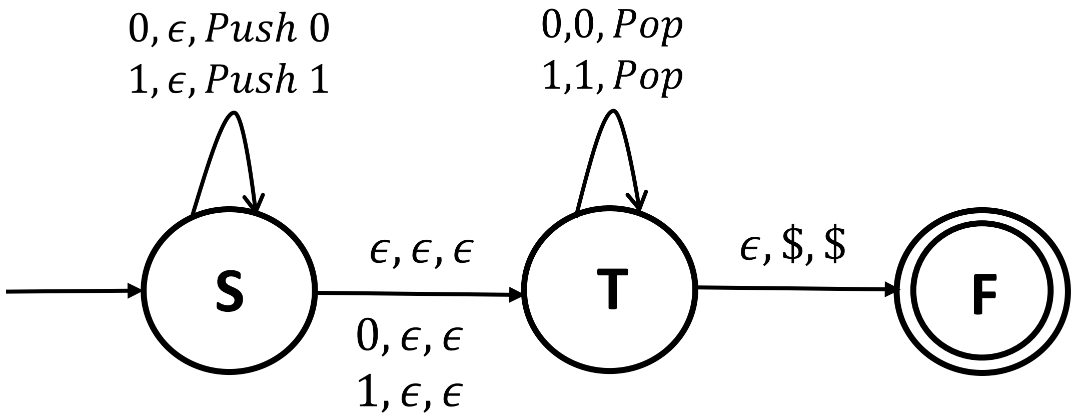
- So determinism is defined slightly differently here. For example: ϵ transitions are allowed, some transitions may be absent (leading to CRASH runs) unlike in the case of DFAs.

- In fact, for a DPDA, the transition rule δ must satisfy the following condition:

For every $q \in Q$, a $\in \Sigma$ and $x \in \Gamma$, **exactly one** of the following values is non-empty

$$\boldsymbol{\delta(q,a,x), \delta(q,a,\epsilon), \delta(q,\epsilon,x)} \textbf{ and } \boldsymbol{\delta(q,\epsilon,\epsilon)}$$

Is this a DPDA? **NO!**



Deterministic Pushdown Automata

DPDAs can be defined in a similar manner to PDAs with the following restriction:

- For each state in the PDA, for any combination of the current input symbol and the current stack symbol **at most** one transition is possible. (At most one valid transition is allowed for any input)

- So determinism is defined slightly differently here. For example: ϵ transitions are allowed, some transitions may be absent (leading to CRASH runs) unlike in the case of DFAs.

- In fact, for a DPDA, the transition rule δ must satisfy the following condition:

For every $q \in Q$, $\mathrm{a} \in \Sigma$ and $x \in \Gamma$, **exactly one** of the following values is non-empty

$$\boldsymbol{\delta(q, a, x), \delta(q, a, \epsilon), \delta(q, \epsilon, x)} \textbf{ and } \boldsymbol{\delta(q, \epsilon, \epsilon)}$$

- Interestingly, the power of DPDAs and PDAs are not the same. It turns out that PDAs are more powerful.

- The language recognized by DPDAs known as **Deterministic CFLs (DCFLs)** are **a proper subset of CFLs**, i.e.

$$\boldsymbol{DCFL \subseteq CFL}$$

Deterministic Pushdown Automata

- For a DPDA, the transition rule δ must satisfy the following condition:

For every $q \in Q$, $\mathrm{a} \in \Sigma$ and $x \in \Gamma$, **exactly one** of the following values is non-empty

$$\boldsymbol{\delta(q, a, x), \delta(q, a, \epsilon), \delta(q, \epsilon, x)} \textbf{ and } \boldsymbol{\delta(q, \epsilon, \epsilon)}$$

- Interestingly, the power of DPDAs and PDAs are not the same. It turns out that PDAs are more powerful.

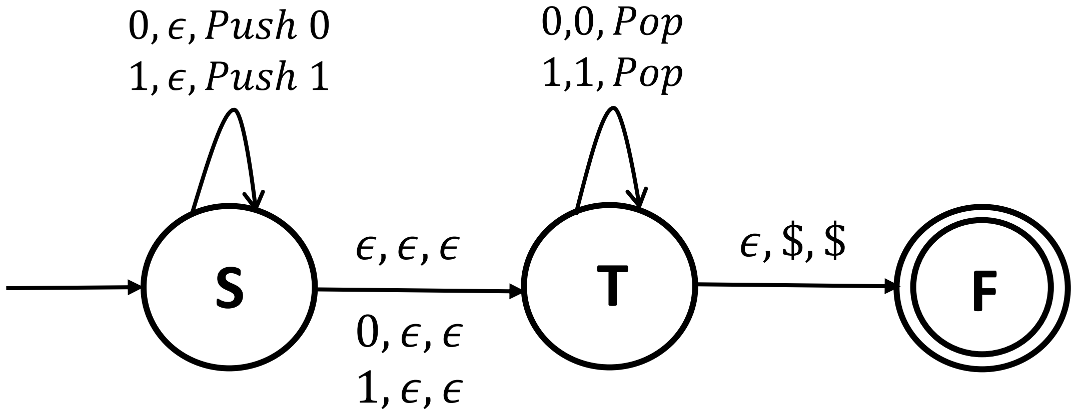
- The language recognized by DPDAs, known as **Deterministic CFLs (DCFLs)** are **a proper subset of CFLs**, i.e.

$$\boldsymbol{DCFL \subseteq CFL}$$

Example: $\boldsymbol{L = \{w|w}$ **is a Palindrome**$\}$

The PDA had to non deterministically guess when half the string has been read and make a transition.

So although $\boldsymbol{L \in CFL, L \notin DCFL}$.

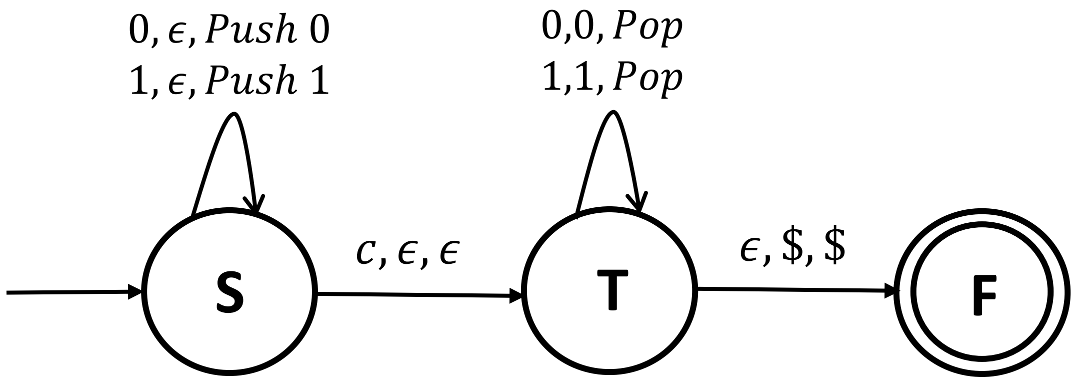


Deterministic Pushdown Automata

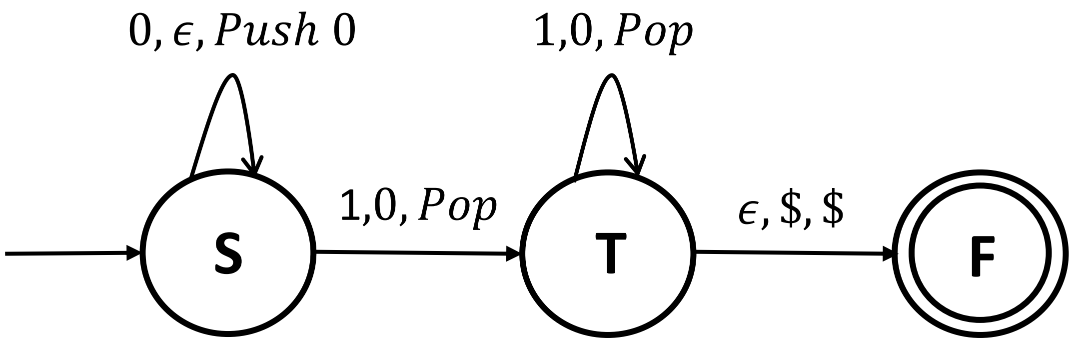
- The language recognized by DPDAs known as **Deterministic CFLs (DCFLs)** are **a proper subset of CFLs**, i.e.

$$DCFL \subseteq CFL$$

- $\boldsymbol{\Sigma = \{0, 1, c\}}$
- $\boldsymbol{L_1 = \{wcw^R | w \in \{0, 1\}^+\}}$
- $\boldsymbol{L_1 \in DCFL}$.



- $\boldsymbol{L_2 = \{0^n 1^n, n \geq 1\}}$
- $\boldsymbol{L_2 \in DCFL}$.



Deterministic Pushdown Automata

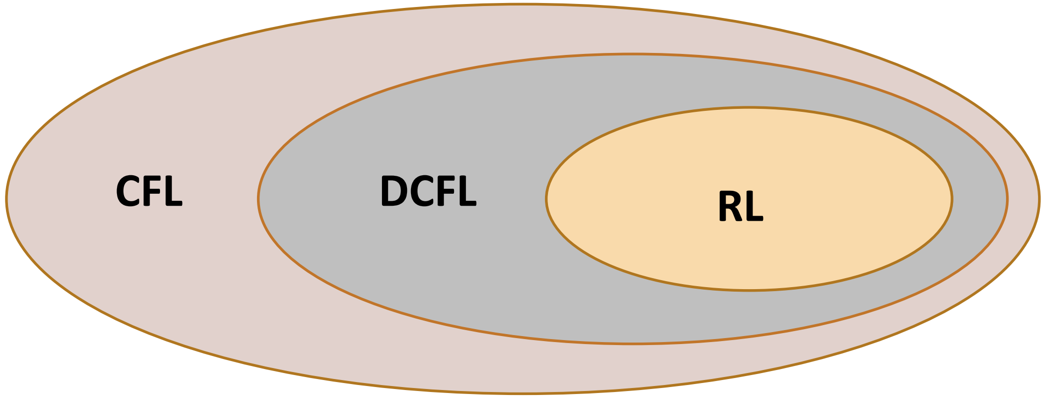
- The language recognized by DPDAs known as **Deterministic CFLs (DCFLs)** are **a proper subset of CFLs**, i.e.

$$\boldsymbol{DCFL \subseteq CFL}$$

$$\boldsymbol{(RL \equiv Regular\ Grammar \equiv Regular\ Expressions \equiv NFA \equiv DFA) \subseteq (DCFL \equiv DPDA) \subseteq (CFL \equiv CFG \equiv PDA)}$$

Next lecture:

- Pumping lemma for CFLs
- Closure properties of CFLs



Thank You!