UMB CS 420

Pushdown Automata (PDAs)

Tuesday, October 18, 2022
Announcements

• HW 4 in
  • Due Sun 10/16 11:59pm EST

• HW 5 out
  • Due Sun 10/23 11:59pm EST

• Reminder: Sean’s Office Hours Monday in-person
  • McCormack 3rd floor 0139
HW2 Review

• $Q' = Q$
• $q'_{\text{start}} = q_{\text{start}}$
• $F' = F$
• $\delta'(q, a) = \{ \delta(q, a) \}$
  • $\forall q \in Q, a \in \Sigma$
• $\delta'(q, \varepsilon) = \{ \}$
  • $\forall q \in Q$

3. Come up with a procedure $\text{DFA} \rightarrow \text{NFA}$ that converts DFAs to equivalent NFAs.

This means that given some DFA $M = (Q, \Sigma, \delta, q_{\text{start}}, F)$ that satisfies the formal definition of DFAs from class, $\text{DFA} \rightarrow \text{NFA}(M)$ should produce some equivalent NFA $N = (Q', \Sigma, \delta', q'_{\text{start}}, F')$ that satisfies the formal definition of NFAs.

A finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where
1. $Q$ is a finite set called the states,
2. $\Sigma$ is a finite set called the alphabet,
3. $\delta : Q \times \Sigma \rightarrow Q$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

A nondeterministic finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where
1. $Q$ is a finite set of states,
2. $\Sigma$ is a finite alphabet,
3. $\delta : Q \times \Sigma \varepsilon \rightarrow \mathcal{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.
HW2 Review

Proof:

**Statement**
1. Assume $B$ and $C$ are reg langs
2. DFA $M$ recognizes $OP(B, C)$
3. $OP(B, C)$ is a regular language
4. $OP$ is closed for reg languages: i.e., if some $B$, $C$ are reg langs, then $OP(B, C)$ is a reg lang

**Justification**
1. Given, from def of closed operation
2. See $M$ construction
3. (2) and Def of regular language
4. From (1) and (3)

(saying “modus ponens” is not a valid justification)
HW2 Review

Proof:

Statement
1. Assume $B$ and $C$ are reg langs
   a) Let $M_B = B$ lang DFA
   b) Let $M_B'$ recognize $\{ x \mid x \notin B \}$
   c) Repeat for $M_C = C$ lang DFA

2. DFA $M$ recognizes $OP(B, C)$

Justification
1. Given, from def of closed operation
   a) Def of reg lang
   b) construct $M_B'$:
      flip accept/non-accept states in $M_B$

2. See $M$ construction
Last Time: Generating Strings with a CFG

\[
G_1 = \\
A \rightarrow 0A1 \\
A \rightarrow B \\
B \rightarrow \#
\]

A CFG represents a context free language!

Strings in CFG’s language = all possible generated strings

\[
L(G_1) = \{0^n#1^n \mid n \geq 0\}
\]

A CFG **generates** a string, by repeatedly applying substitution rules:

\[
A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000B111 \Rightarrow 000#111
\]

Start variable

Stop when string is all terminals
Last Time:

<table>
<thead>
<tr>
<th><strong>Regular Languages</strong></th>
<th><strong>Context-Free Languages (CFLs)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular Expression</td>
<td>Context-Free Grammar (CFG)</td>
</tr>
<tr>
<td>A Reg Expr   <strong>describes</strong> a Regular lang</td>
<td>A CFG <strong>describes</strong> a CFL</td>
</tr>
</tbody>
</table>
Today:

<table>
<thead>
<tr>
<th>Regular Languages</th>
<th>Context-Free Languages (CFLs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular Expression</td>
<td>Context-Free Grammar (CFG)</td>
</tr>
<tr>
<td>A Reg Expr describes a Regular lang</td>
<td>A CFG describes a CFL</td>
</tr>
</tbody>
</table>

**TODAY:**

<table>
<thead>
<tr>
<th>Finite Automaton (FSM)</th>
<th>Push-down automaton (PDA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>An FSM recognizes a Regular lang</td>
<td>A PDA recognizes a CFL</td>
</tr>
</tbody>
</table>
Today:

<table>
<thead>
<tr>
<th>Regular Languages</th>
<th>Context-Free Languages (CFLs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular Expression</td>
<td>Context-Free Grammar (CFG)</td>
</tr>
<tr>
<td>A Reg Expr describes a Regular lang</td>
<td>A CFG describes a CFL</td>
</tr>
</tbody>
</table>

**TODAY:**

<table>
<thead>
<tr>
<th>Finite Automaton (FSM)</th>
<th>Push-down automaton (PDA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>An FSM recognizes a Regular lang</td>
<td>A PDA recognizes a CFL</td>
</tr>
</tbody>
</table>

**KEY DIFFERENCE:**

<table>
<thead>
<tr>
<th>A Regular lang is defined with a FSM</th>
<th>A CFL is defined with a CFG</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Must prove: Reg Expr ⇔ Reg lang</em></td>
<td><em>Must prove: PDA ⇔ CFL</em></td>
</tr>
</tbody>
</table>
Pushdown Automata (PDA)

\[ \text{PDA} = \text{NFA} + \text{a stack} \]
What is a Stack?

- A **restricted** kind of (infinite) memory
- Access to top element only
- 2 Operations only: **push**, **pop**
Pushdown Automata (PDA)

- **PDA = NFA + a stack**
  - Infinite memory
  - Can only read/write top location
    - Push/pop
An Example PDA

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

$q_1 \xrightarrow{\varepsilon, \varepsilon} \$ \rightarrow q_2$

$q_2 \xrightarrow{0, \varepsilon} 0 \rightarrow q_2$

$q_2 \xrightarrow{\text{Pop} \ 0} q_3$

$q_3 \xrightarrow{1, 0} \varepsilon \rightarrow q_3$

$q_3 \xrightarrow{\text{Pop} \ 0} q_4$

$q_4 \xrightarrow{\varepsilon, \$} \varepsilon \rightarrow q_4$

Read input: no
No Pop
Push
Read 0
No Pop
Push 0

$\$ = special symbol, indicating empty stack

Can only pop this (and accept) when stack is empty, i.e., when # 0s matches # 1s

(and repeat)
A **pushdown automaton** is a 6-tuple \((Q, \Sigma, \Gamma, \delta, q_0, F)\), where \(Q\), \(\Sigma\), \(\Gamma\), and \(F\) are all finite sets, and

1. \(Q\) is the set of states,
2. \(\Sigma\) is the input alphabet,
3. \(\Gamma\) is the stack alphabet,
4. \(\delta : Q \times \Sigma_{\varepsilon} \times \Gamma_{\varepsilon} \rightarrow \mathcal{P}(Q \times \Gamma_{\varepsilon})\) is the transition function,
5. \(q_0 \in Q\) is the start state, and
6. \(F \subseteq Q\) is the set of accept states.

Stack alphabet can have special stack symbols, e.g., $.

Non-deterministic: produces a set of \((\text{State, Stack Char})\) pairs.
PDA Formal Definition Example

\[
Q = \{q_1, q_2, q_3, q_4\}, \\
\Sigma = \{0, 1\}, \\
\Gamma = \{0, $\}, \\
F = \{q_1, q_4\},
\]

A **pushdown automaton** is a 6-tuple \((Q, \Sigma, \Gamma, \delta, q_0, F)\), where \(Q, \Sigma, \Gamma,\) and \(F\) are all finite sets, and

1. \(Q\) is the set of states,
2. \(\Sigma\) is the input alphabet,
3. \(\Gamma\) is the stack alphabet,
4. \(\delta: Q \times \Sigma \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma)\) is the transition function,
5. \(q_0 \in Q\) is the start state, and
6. \(F \subseteq Q\) is the set of accept states.
\( Q = \{ q_1, q_2, q_3, q_4 \}, \)
\( \Sigma = \{ 0, 1 \}, \)
\( \Gamma = \{ 0, \$ \}, \)
\( F = \{ q_1, q_4 \}, \) and

\( \delta \) is given by the following table, wherein blank entries signify \( \emptyset \).

<table>
<thead>
<tr>
<th>Stack: ( $ )</th>
<th>Input: 0</th>
<th>( \varepsilon )</th>
<th>Input: 1</th>
<th>( \varepsilon )</th>
<th>Pop: ( \varepsilon )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>1</td>
<td>( {(q_2, 0)} )</td>
<td>( {(q_3, \varepsilon)} )</td>
<td>( {(q_2, \varepsilon)} )</td>
<td>( {(q_2, \varepsilon)} )</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>( {(q_2, \varepsilon)} )</td>
<td>( {(q_3, \varepsilon)} )</td>
<td>( {(q_2, \varepsilon)} )</td>
<td>( {(q_2, \varepsilon)} )</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>( {(q_2, \varepsilon)} )</td>
<td>( {(q_2, \varepsilon)} )</td>
<td>( {(q_2, \varepsilon)} )</td>
<td>( {(q_2, \varepsilon)} )</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>( {(q_2, \varepsilon)} )</td>
<td>( {(q_2, \varepsilon)} )</td>
<td>( {(q_2, \varepsilon)} )</td>
<td>( {(q_2, \varepsilon)} )</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>( {(q_2, \varepsilon)} )</td>
<td>( {(q_2, \varepsilon)} )</td>
<td>( {(q_2, \varepsilon)} )</td>
<td>( {(q_2, \varepsilon)} )</td>
<td></td>
</tr>
</tbody>
</table>

A pushdown automaton is a 6-tuple \((Q, \Sigma, \Gamma, \delta, q_0, F)\), where \( Q, \Sigma, \Gamma, \) and \( F \) are all finite sets, and

1. \( Q \) is the set of states,
2. \( \Sigma \) is the input alphabet,
3. \( \Gamma \) is the stack alphabet,
4. \( \delta: Q \times \Sigma_e \times \Gamma_e \rightarrow \mathcal{P}(Q \times \Gamma_e) \) is the transition function,
5. \( q_0 \in Q \) is the start state, and
6. \( F \subseteq Q \) is the set of accept states.
$Q = \{q_1, q_2, q_3, q_4\}$,
$\Sigma = \{0, 1\}$,
$\Gamma = \{0, \$\}$,
$F = \{q_1, q_4\}$, and

$\delta$ is given by the following table, wherein blank entries signify $\emptyset$.

<table>
<thead>
<tr>
<th>Input:</th>
<th>0</th>
<th>$\varepsilon$</th>
<th>1</th>
<th>$\varepsilon$</th>
<th>$\varepsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack:</td>
<td>0</td>
<td>$$</td>
<td>$\varepsilon$</td>
<td>0</td>
<td>$$</td>
</tr>
<tr>
<td>$q_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$q_2$</td>
<td></td>
<td></td>
<td>${(q_2, 0)}$</td>
<td>${(q_3, \varepsilon)}$</td>
<td>${(q_3, \varepsilon)}$</td>
</tr>
<tr>
<td>$q_3$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$q_4$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A pushdown automaton is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where $Q$, $\Sigma$, $\Gamma$, and $F$ are all finite sets, and

1. $Q$ is the set of states,
2. $\Sigma$ is the input alphabet,
3. $\Gamma$ is the stack alphabet,
4. $\delta : Q \times \Sigma \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma)$ is the transition function,
5. $q_0 \in Q$ is the start state, and
6. $F \subseteq Q$ is the set of accept states.
$Q = \{q_1, q_2, q_3, q_4\}$,
$\Sigma = \{0, 1\}$,
$\Gamma = \{0, \$\}$,
$F = \{q_1, q_4\}$, and

$\delta$ is given by the following table, wherein blank entries signify $\emptyset$.

<table>
<thead>
<tr>
<th>Input</th>
<th>Stack</th>
<th>0</th>
<th>$$</th>
<th>$\varepsilon$</th>
<th>1</th>
<th>$$</th>
<th>$\varepsilon$</th>
<th>$\varepsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack</td>
<td>$q_1$</td>
<td>${(q_2, 0)}$</td>
<td>${(q_3, \varepsilon)}$</td>
<td>${(q_4, \varepsilon)}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stack</td>
<td>$q_2$</td>
<td>${(q_3, \varepsilon)}$</td>
<td>${(q_4, \varepsilon)}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stack</td>
<td>$q_3$</td>
<td>${(q_4, \varepsilon)}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stack</td>
<td>$q_4$</td>
<td>${(q_2, $)}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A **pushdown automaton** is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where $Q, \Sigma, \Gamma$, and $F$ are all finite sets, and

1. $Q$ is the set of states,
2. $\Sigma$ is the input alphabet,
3. $\Gamma$ is the stack alphabet,
4. $\delta: Q \times \Sigma \times \Gamma \rightarrow P(Q \times \Gamma)$ is the transition function,
5. $q_0 \in Q$ is the start state, and
6. $F \subseteq Q$ is the set of accept states.
\[ Q = \{q_1, q_2, q_3, q_4\}, \]
\[ \Sigma = \{0, 1\}, \]
\[ \Gamma = \{0, \$$\}, \]
\[ F = \{q_1, q_4\}, \]
and
\[ \delta \] is given by the following table, wherein blank entries signify \( \emptyset \).

<table>
<thead>
<tr>
<th>Input:</th>
<th>0</th>
<th>( \varepsilon )</th>
<th>1</th>
<th>( \varepsilon )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack:</td>
<td>$</td>
<td>( \varepsilon )</td>
<td>0</td>
<td>( \varepsilon )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>0</td>
<td>( {q_2, 0})</td>
<td>2</td>
<td>( {q_4, \varepsilon})</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( {q_3, \varepsilon})</td>
<td>3</td>
<td>( {q_4, \varepsilon})</td>
<td></td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( {q_3, \varepsilon})</td>
<td>4</td>
<td>( {(q_2, $$)})</td>
<td></td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( \varepsilon ), ( \varepsilon \rightarrow $</td>
<td>5</td>
<td>( \varepsilon )</td>
<td></td>
</tr>
</tbody>
</table>

A pushdown automaton is a 6-tuple \((Q, \Sigma, \Gamma, \delta, q_0, F)\), where \(Q, \Sigma, \Gamma,\) and \(F\) are all finite sets, and

1. \(Q\) is the set of states,
2. \(\Sigma\) is the input alphabet,
3. \(\Gamma\) is the stack alphabet,
4. \(\delta: Q \times \Sigma \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma)\) is the transition function,
5. \(q_0 \in Q\) is the start state, and
6. \(F \subseteq Q\) is the set of accept states.
\[ Q = \{ q_1, q_2, q_3, q_4 \}, \]
\[ \Sigma = \{ 0, 1 \}, \]
\[ \Gamma = \{ 0, \$ \}, \]
\[ F = \{ q_1, q_4 \}, \]

\( \delta \) is given by the following table, wherein blank entries signify \( \emptyset \).

<table>
<thead>
<tr>
<th>Input:</th>
<th>0</th>
<th>1</th>
<th>( \varepsilon )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack:</td>
<td>0</td>
<td>0</td>
<td>( \varepsilon )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>{ (q_2, 0) }</td>
<td>{ (q_3, \varepsilon) }</td>
<td>2</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>1</td>
<td>{ (q_3, \varepsilon) }</td>
<td>3</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>{ (q_4, \varepsilon) }</td>
<td>{ (q_2, $) }</td>
<td>5</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>\emptyset</td>
<td>\emptyset</td>
<td>\emptyset</td>
</tr>
</tbody>
</table>

A **pushdown automaton** is a 6-tuple \( (Q, \Sigma, \Gamma, \delta, q_0, F) \), where \( Q, \Sigma, \Gamma, \) and \( F \) are all finite sets, and

1. \( Q \) is the set of states,
2. \( \Sigma \) is the input alphabet,
3. \( \Gamma \) is the stack alphabet,
4. \( \delta: Q \times \Sigma \times \Gamma \rightarrow P(Q \times \Gamma) \) is the transition function,
5. \( q_0 \in Q \) is the start state, and
6. \( F \subseteq Q \) is the set of accept states.
In-class exercise:
Fill in the blanks

\[ Q = \]

\[ \Sigma = \]

\[ \Gamma = \]

\[ F = \]

\( \delta \) is given by the following table, wherein blank entries signify \( \emptyset \).

<table>
<thead>
<tr>
<th>Input</th>
<th>0</th>
<th>1</th>
<th>( \varepsilon )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack</td>
<td>0</td>
<td>$</td>
<td>\varepsilon</td>
</tr>
</tbody>
</table>

PDA \( M_3 \) recognizing the language \( \{ w w^R \mid w \in \{0, 1\}^* \} \)

A pushdown automaton is a 6-tuple \( (Q, \Sigma, \Gamma, \delta, q_0, F) \), where \( Q, \Sigma, \Gamma, \) and \( F \) are all finite sets, and

1. \( Q \) is the set of states,
2. \( \Sigma \) is the input alphabet,
3. \( \Gamma \) is the stack alphabet,
4. \( \delta: Q \times \Sigma_e \times \Gamma_e \rightarrow \mathcal{P}(Q \times \Gamma_e) \) is the transition function,
5. \( q_0 \in Q \) is the start state, and
6. \( F \subseteq Q \) is the set of accept states.
In-class exercise: Fill in the blanks

$Q = \{q_1, q_2, q_3, q_4\}$,

$\Sigma = \{0,1\}$,

$\Gamma = \{0,1,\$\}$,

$F = \{q_4\}$

$\delta$ is given by the following table, wherein blank entries signify $\emptyset$.

<table>
<thead>
<tr>
<th>Input:</th>
<th>0</th>
<th>1</th>
<th>$\varepsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack:</td>
<td>0</td>
<td>$$</td>
<td>$\varepsilon$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>${(q_2, 0)}$</td>
<td>${(q_2, 1)}$</td>
<td>${(q_2, $), $(q_3, \varepsilon)}$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>${(q_3, \varepsilon)}$</td>
<td>${(q_3, \varepsilon)}$</td>
<td>${(q_4, \varepsilon)}$</td>
</tr>
</tbody>
</table>

PDA $M_3$ recognizing the language $\{ww^R | w \in \{0,1\}^*\}$
**Flashback:** DFA Computation Model

<table>
<thead>
<tr>
<th>Informally</th>
<th>Formally (i.e., mathematically)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• “Program” = a finite automata</td>
<td>• $M = (Q, \Sigma, \delta, q_0, F)$</td>
</tr>
<tr>
<td>• Input = string of chars, e.g. “1101”</td>
<td>• $w = w_1w_2 \cdots w_n$</td>
</tr>
<tr>
<td><strong>To run a “program”:</strong></td>
<td></td>
</tr>
<tr>
<td>• Start in “start state”</td>
<td>• $r_0 = q_0$</td>
</tr>
<tr>
<td>• <strong>Repeat:</strong></td>
<td>• $r_i = \delta(r_{i-1}, w_i)$, for $i = 1, \ldots, n$</td>
</tr>
<tr>
<td>• Read 1 char;</td>
<td></td>
</tr>
<tr>
<td>• Change state according to the <strong>transition</strong> table</td>
<td>widening the result to allow for an infinite sequence of states</td>
</tr>
<tr>
<td>• Result =</td>
<td>• $M$ accepts $w$ if</td>
</tr>
<tr>
<td>• “Accept” if last state is “Accept” state</td>
<td>sequence of states $r_0, r_1, \ldots, r_n$ in $Q$ exists ...</td>
</tr>
<tr>
<td>• “Reject” otherwise</td>
<td>with $r_n \in F$</td>
</tr>
</tbody>
</table>

**A sequence of states** represents a DFA computation
PDA Configurations (IDs)

• A **configuration** (or **ID**) is a “**snapshot**” of a PDA’s computation

• 3 components \((q, w, \gamma)\):
  - \(q\) = the current state
  - \(w\) = the remaining input string
  - \(\gamma\) = the stack contents

• A **sequence of configurations** represents a **PDA** computation
**Flashback:** A DFA Extended Transition Fn

Define **extended transition function:**

- **Domain:**
  - Beginning state $q \in Q$ (not necessarily the start state)
  - Input string $w = w_1 w_2 \cdots w_n$ where $w_i \in \Sigma$

- **Range:**
  - Ending state (not necessarily an accept state)

(Defined recursively)

- **Base case:** $\hat{\delta}(q, \varepsilon) = q$

- **Recursive case:** $\hat{\delta}(q, w) = \hat{\delta}(\delta(q, w_1), w_2 \cdots w_n)$

$\hat{\delta}: Q \times \Sigma^* \rightarrow Q$ is the transition function

This specifies the sequence of states representing a DFA computation.
PDA Computation, Formally

\[ P = (Q, \Sigma, \Gamma, \delta, q_0, F) \]

**Single-step**

Before / After configurations

\[(q_1, aw, X\beta) \vdash (q_2, w, \alpha\beta)\]

- Read Input
- Pop
- Push

if \(\delta(q_1, a, X)\) contains \((q_2, \alpha)\)

- \(q_1, q_2 \in Q\)
- \(a \in \Sigma\)
- \(w \in \Sigma^*\)
- \(X \in \Gamma\)
- \(\beta, \alpha \in \Gamma^*\)

**Extended**

- **Base Case**
  \(I \vdash^* I\) for any ID \(I\)

- **Recursive Case**
  \(I \vdash^* J\) if there exists some ID \(K\) such that \(I \vdash K\) and \(K \vdash^* J\)

---

A configuration \((q, w, \gamma)\) has three components

- \(q\) = the current state
- \(w\) = the remaining input string
- \(\gamma\) = the stack contents
PDA Running Input String Example

\((q_1, 0011, \varepsilon)\)
PDA Running Input String Example

$$(q_1, 0011, \varepsilon) \xrightarrow{0, \varepsilon \rightarrow 0} (q_2, 0011, \$) \xrightarrow{1,0 \rightarrow \varepsilon} (q_2, 011, 0\$)$$

Read 0, push 0
PDA Running Input String Example

\[(q_1, 0011, \varepsilon) \vdash (q_2, 0011, \$$) \]
\[\vdash (q_2, 011, 0\$$) \]
\[\vdash (q_2, 11, 00\$$) \]
PDA Running Input String Example

\[(q_1, 0011, \varepsilon) \Rightarrow (q_2, 0011, \$)\]
\[\Rightarrow (q_2, 011, 0\$)\]
\[\Rightarrow (q_2, 11, 00\$)\]
\[\Rightarrow (q_3, 1, 0\$)\]

Read 1, pop 0
PDA Running Input String Example

$$(q_1, 0011, \varepsilon) \vdash (q_2, 0011, \$$)
$$\vdash (q_2, 011, 0\$$)
$$\vdash (q_2, 11, 00\$$)
$$\vdash (q_3, 1, 0\$$)
$$\vdash (q_3, \varepsilon, \$$)

Read 1, pop 0
PDA Running Input String Example

\[(q_1, 0011, \varepsilon) \vdash (q_2, 0011, \$$)\]
\[\vdash (q_2, 011, 0\$$)\]
\[\vdash (q_2, 11, 00\$$)\]
\[\vdash (q_3, 1, 0\$$)\]
\[\vdash (q_3, \varepsilon, \$$)\]
\[\vdash (q_4, \varepsilon, \varepsilon)\]
Flashback: Computation and Languages

- The **language** of a machine is the set of all strings that it accepts.

- E.g., An FSM $M$ accepts $w$ if $\hat{\delta}(q_0, w) \in F$.

- Language of $M = L(M) = \{ w \mid M \text{ accepts } w \}$.
Language of a PDA

\[ P = (Q, \Sigma, \Gamma, \delta, q_0, F) \]

\[ L(P) = \{ w \mid (q_0, w, \varepsilon) \vdash^* (q, \varepsilon, \alpha) \} \text{ where } q \in F \]

A configuration \((q, w, \gamma)\) has three components:
- \(q\) = the current state
- \(w\) = the remaining input string
- \(\gamma\) = the stack contents
Pushdown Automata (PDA)

• **PDA** = NFA + a stack
  • Infinite memory
  • Can only read/write top location: Push/pop

• **Want to prove**: PDAs represent CFLs!

• **We know**: a CFL, by definition, is a language that is generated by a CFG

• **Need to show**: PDA $\Leftrightarrow$ CFG

• Then, **to prove** that a language is a CFL, we can either:
  • Create a CFG, or
  • Create a PDA
A lang is a CFL iff some PDA recognizes it

⇒ If a language is a CFL, then a PDA recognizes it
  • (Easier)
  • **We know**: A CFL has a CFG describing it *(definition of CFL)*
  • **Must show**: the CFG has an equivalent PDA

⇐ If a PDA recognizes a language, then it’s a CFL
Shorthand: Multi-Symbol Stack Pushes

Note the reverse order of pushes
CFG → PDA (sketch)

- Construct a PDA from CFG such that:
  - PDA accepts input string only if the CFG can generate that string
- Intuitively, PDA will nondeterministically try all rules
CFG → PDA (sketch)

• Construct a PDA from CFG such that:
  • PDA accepts input string only if the CFG can generate that string

• Intuitively, PDA will nondeterministically try all rules

- Transition diagram:
  - Start state: $q_{start}$
  - Transition: $\epsilon, \epsilon \rightarrow S$ (push start variable onto stack)
  - Transition: $\epsilon, A \rightarrow w$ for rule $A \rightarrow w$
  - Transition: $a, a \rightarrow \epsilon$ for terminal $a$
  - Accept state: $q_{accept}$

- Transition rules:
  - If stack top is a variable $A$, pop it and (nondeterministically) push rule’s right-sides
  - If stack top is a terminal $a$, pop it and read matching input
Example **CFG→PDA**

Transition Rules:

- $S \rightarrow aTb \mid b$
- $T \rightarrow Ta \mid \varepsilon$

States and Transitions:

1. **$q_{start}$**
   - $\varepsilon, \varepsilon \rightarrow S$
   - $\varepsilon, \varepsilon \rightarrow \varepsilon$
   - $\varepsilon, \varepsilon \rightarrow \varepsilon$

2. **$q_{loop}$**
   - $\varepsilon, S \rightarrow b$
   - $\varepsilon, T \rightarrow a$
   - $\varepsilon, T \rightarrow \varepsilon$
   - $a, a \rightarrow \varepsilon$
   - $b, b \rightarrow \varepsilon$

3. **$q_{accept}$**

Transition Arrows:

- Arrows indicate transitions between states.
- Arrows also show transitions based on input symbols $a$ and $b$.

Stack Operations:

- If stack top is **variable $S$**, pop $S$ and push rule right-sides (in reverse order).
Example **CFG→PDA**

- **Production Rules:**
  - $S \rightarrow aTb \mid b$
  - $T \rightarrow Ta \mid \epsilon$

- **Transition Diagram:**
  - $q_{start}$
  - $q_{loop}$
  - $q_{accept}$

- **Transitions:**
  - $\epsilon, \epsilon \rightarrow \$ (from $q_{start}$ to $q_{loop}$)
  - $\epsilon, \epsilon \rightarrow S$ (from $q_{loop}$ to itself)
  - $\epsilon, \$ \rightarrow \epsilon$ (from $q_{loop}$ to $q_{accept}$)
  - $\epsilon, S \rightarrow b$ (from $q_{loop}$ to $\epsilon, \epsilon \rightarrow T$
  - $\epsilon, T \rightarrow a$ (from $q_{loop}$ to $\epsilon, \epsilon \rightarrow T$
  - $\epsilon, T \rightarrow \epsilon$ (from $q_{loop}$ to $\epsilon, \epsilon \rightarrow T$
  - $\epsilon, S \rightarrow b$ (from $q_{loop}$ to $\epsilon, \epsilon \rightarrow a$
  - $a, a \rightarrow \epsilon$ (from $q_{loop}$ to $q_{accept}$)
  - $b, b \rightarrow \epsilon$ (from $q_{loop}$ to $q_{accept}$)
Example **CFG→PDA**

\[ S \rightarrow aTb \mid b \]
\[ T \rightarrow Ta \mid \varepsilon \]

Diagram:

- **q_{start}**: States start here.
- **q_{loop}**: States in the middle of the loop.
- **q_{accept}**: States for accepting.

Transition rules:
- \( \varepsilon, \varepsilon \rightarrow \$ \)
- \( \varepsilon, \varepsilon \rightarrow S \)
- \( \varepsilon, S \rightarrow b \)
- \( \varepsilon, T \rightarrow a \)
- \( \varepsilon, \varepsilon \rightarrow T \)
- \( \varepsilon, S \rightarrow b \)
- \( \varepsilon, T \rightarrow \varepsilon \)
- \( a, a \rightarrow \varepsilon \)
- \( b, b \rightarrow \varepsilon \)

Legend:
- If stack top is a terminal, pop and read matching input.

Note: The diagram shows a context-free grammar (CFG) converted to a pushdown automaton (PDA).
Example **CFG→PDA**

**Example Derivation using CFG:**
- $S \Rightarrow aTa$ (using rule $S \rightarrow aTa$)
- $\Rightarrow aTab$ (using rule $T \rightarrow Ta$)
- $\Rightarrow aab$ (using rule $T \rightarrow \epsilon$)

**PDA Example**

<table>
<thead>
<tr>
<th>State</th>
<th>Input</th>
<th>Stack</th>
<th>Equiv Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_{start}$</td>
<td>aab</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$q_{loop}$</td>
<td>aab</td>
<td>$S$$</td>
<td>$S \rightarrow aTa$</td>
</tr>
<tr>
<td>$q_{loop}$</td>
<td>aab</td>
<td>$aTa$</td>
<td>$S \rightarrow aTa$</td>
</tr>
<tr>
<td>$q_{loop}$</td>
<td>ab</td>
<td>$7b$$</td>
<td>$T \rightarrow Ta$</td>
</tr>
<tr>
<td>$q_{loop}$</td>
<td>ab</td>
<td>$Tab$$</td>
<td>$T \rightarrow Ta$</td>
</tr>
<tr>
<td>$q_{loop}$</td>
<td>ab</td>
<td>$ab$$</td>
<td>$T \rightarrow \epsilon$</td>
</tr>
<tr>
<td>$q_{loop}$</td>
<td>b</td>
<td>$b$$</td>
<td></td>
</tr>
<tr>
<td>$q_{accept}$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example \textbf{CFG}→\textbf{PDA}

\begin{align*}
S & \rightarrow aTb \quad \text{(using rule } S \rightarrow aTb) \\
& \Rightarrow aTab \quad \text{(using rule } T \rightarrow Ta) \\
& \Rightarrow aab \quad \text{(using rule } T \rightarrow \epsilon) \\
S & \rightarrow aTb \mid b \\
T & \rightarrow Ta \mid \epsilon
\end{align*}

If stack top is \textbf{variable} $S$, pop $S$ and push rule right-sides (in rev order)

\begin{tabular}{|c|c|c|c|}
\hline
\textbf{State} & \textbf{Input} & \textbf{Stack} & \textbf{Equiv Rule} \\
\hline
$q_{\text{start}}$ & ab &  &  \\
$q_{\text{loop}}$ & ab & aTb$\$ & $S \rightarrow aTb$ \\
$q_{\text{loop}}$ & ab & $Tb\$ &  \\
$q_{\text{loop}}$ & ab & T$\$ &  \\
$q_{\text{loop}}$ & ab & ab$\$ & $T \rightarrow Ta$ \\
$q_{\text{loop}}$ & ab & ab$\$ & $T \rightarrow \epsilon$ \\
$q_{\text{loop}}$ & b & b$\$ &  \\
$q_{\text{loop}}$ &  & $\$ &  \\
$q_{\text{accept}}$ &  &  &  \\
\hline
\end{tabular}
Example **CFG → PDA**

**Example Derivation using CFG:**

\[ S \rightarrow aTb \] (using rule \( S \rightarrow aTb \))

\[ \Rightarrow aTab \] (using rule \( T \rightarrow Ta \))

\[ \Rightarrow aab \] (using rule \( T \rightarrow \varepsilon \))

### PDA Example

<table>
<thead>
<tr>
<th>State</th>
<th>Input</th>
<th>Stack</th>
<th>Equiv Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_{start} )</td>
<td>aab</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( q_{loop} )</td>
<td>aab</td>
<td>$$</td>
<td>( S \rightarrow aTb )</td>
</tr>
<tr>
<td>( q_{loop} )</td>
<td>aab</td>
<td>aTb$$</td>
<td></td>
</tr>
<tr>
<td>( q_{loop} )</td>
<td>ab</td>
<td>7b$$</td>
<td></td>
</tr>
<tr>
<td>( q_{loop} )</td>
<td>ab</td>
<td>Tab$$</td>
<td>( T \rightarrow Ta )</td>
</tr>
<tr>
<td>( q_{loop} )</td>
<td>ab</td>
<td>ab$$</td>
<td>( T \rightarrow \varepsilon )</td>
</tr>
<tr>
<td>( q_{loop} )</td>
<td>b</td>
<td>b$$</td>
<td></td>
</tr>
<tr>
<td>( q_{accept} )</td>
<td></td>
<td>$$</td>
<td></td>
</tr>
</tbody>
</table>
Example **CFG→PDA**

---

**Example Derivation using CFG:**

\[ S \rightarrow aTb \] (using rule \( S \rightarrow aTb \))

\[ \rightarrow aTab \] (using rule \( T \rightarrow Ta \))

\[ \rightarrow aab \] (using rule \( T \rightarrow \varepsilon \))

---

**PDA Example**

<table>
<thead>
<tr>
<th>State</th>
<th>Input</th>
<th>Stack</th>
<th>Equiv Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_{start} )</td>
<td>aab</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( q_{loop} )</td>
<td>aab</td>
<td>$S$</td>
<td></td>
</tr>
<tr>
<td>( q_{loop} )</td>
<td>aab</td>
<td>aTb$</td>
<td>( S \rightarrow aTb )</td>
</tr>
<tr>
<td>( q_{loop} )</td>
<td>ab</td>
<td>Tb$</td>
<td></td>
</tr>
<tr>
<td>( q_{loop} )</td>
<td>ab</td>
<td>Tab$</td>
<td>( T \rightarrow Ta )</td>
</tr>
<tr>
<td>( q_{loop} )</td>
<td>ab</td>
<td>ab$</td>
<td>( T \rightarrow \varepsilon )</td>
</tr>
<tr>
<td>( q_{loop} )</td>
<td>b</td>
<td>b$</td>
<td></td>
</tr>
<tr>
<td>( q_{accept} )</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A lang is a CFL iff some PDA recognizes it

⇒ If a language is a CFL, then a PDA recognizes it
  • Convert CFG→PDA

⇐ If a PDA recognizes a language, then it’s a CFL
  • (Harder)
  • Must Show: PDA has an equivalent CFG
PDA→CFG: Prelims

Before converting PDA to CFG, modify it so:

1. It has a single accept state, $q_{\text{accept}}$.
2. It empties its stack before accepting.
3. Each transition either pushes a symbol onto the stack (a push move) or pops one off the stack (a pop move), but it does not do both at the same time.

Important:
This doesn’t change the language recognized by the PDA
PDA $P \rightarrow$ CFG $G$: Variables

$P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{\text{accept}}\})$  
variables of $G$ are $\{A_{pq} \mid p, q \in Q\}$

- **Want:** if $P$ goes from state $p$ to $q$ reading input $x$, then some $A_{pq}$ generates $x$

- **So:** For every pair of states $p, q$ in $P$, add variable $A_{pq}$ to $G$

- **Then:** connect the variables together by,
  - Add rules: $A_{pq} \rightarrow A_{pr}A_{rq}$, for each state $r$
  - These rules allow grammar to simulate every possible transition
  - (We haven’t added input read/generated terminals yet)

- **To add terminals:** pair up stack pushes and pops (essence of a CFL)
PDA $P \rightarrow$ CFG $G$: Generating Strings

$P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{\text{accept}}\})$

variables of $G$ are $\{A_{pq} \mid p, q \in Q\}$

- **The key**: pair up stack pushes and pops (essence of a CFL)

  if $\delta(p, a, \varepsilon)$ contains $(r, u)$ and $\delta(s, b, u)$ contains $(q, \varepsilon)$,

  put the rule $A_{pq} \rightarrow aA_{rs}b$ in $G$
PDA $P \rightarrow$ CFG $G$: Generating Strings

$P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{\text{accept}}\})$

variables of $G$ are $\{A_{pq} | p, q \in Q\}$

- The key: pair up stack pushes and pops (essence of a CFL)

if $\delta(p, a, \varepsilon)$ contains $(r, u)$ and $\delta(s, b, u)$ contains $(q, \varepsilon)$,

put the rule $A_{pq} \rightarrow aA_{rs}b$ in $G$
PDA $P \rightarrow$ CFG $G$: Generating Strings

$P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{accept}\})$  
variables of $G$ are $\{A_{pq} | p, q \in Q\}$

- **The key**: pair up stack pushes and pops (essence of a CFL)

if $\delta(p, a, \varepsilon)$ contains $(r, u)$ and $\delta(s, b, u)$ contains $(q, \varepsilon)$,

put the rule $A_{pq} \rightarrow aA_{rs}b$ in $G$
A language is a CFL $\iff$ A PDA recognizes it

$\checkmark$ $\Rightarrow$ If a language is a CFL, then a PDA recognizes it
  • Convert CFG$\Rightarrow$PDA

$\checkmark$ $\Leftarrow$ If a PDA recognizes a language, then it’s a CFL
  • Convert PDA$\Rightarrow$CFG
Check-in Quiz 10/18

On Gradescope