UMB CS 622

PDA Computation

Friday, March 22, 2024
Announcements

• HW 5 out
  • Due Mon 3/25 12pm noon
Pushdown Automata (PDA)

• **PDA** = NFA + a stack
  • Infinite memory
  • push/pop top location only
An Example PDA

A PDA transition has 3 parts:
- Read
- Pop
- Push

Read (no) input
No Pop
Push
Read 0
No Pop
Push 0

$q_1 \xrightarrow{\varepsilon, \varepsilon} \$$(and repeat)

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

$q_3 \xrightarrow{1, 0} \varepsilon$ (and repeat)

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

This machine can only pop $\$$(and accept) when stack is empty, i.e., when \# 0s = \# 1s

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

Last Time:
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.

Stack alphabet has special stack symbols, e.g., $\$$. 

Non-deterministic! Result of a step is set of (State, Stack Char) pairs.
PDA Formal Definition Example

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

Stack alphabet has special stack symbol $\$

A pushdown automaton is a 6-tuple \((Q, \Sigma, \Gamma, \delta, q_0, F)\), where \(Q\), \(\Sigma\), \(\Gamma\), \(\delta\), \(q_0\), 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>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack: 0</td>
<td>$$</td>
<td>$\varepsilon$</td>
<td>0</td>
<td>$$</td>
<td>$\varepsilon$</td>
<td>0</td>
<td>$$</td>
<td>$\varepsilon$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>${(q_2, 0)}$</td>
<td>${(q_3, \varepsilon)}$</td>
<td>${(q_3, \varepsilon)}$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>$q_2$</td>
<td>${(q_4, \varepsilon)}$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>1</td>
<td>$\emptyset$</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. $\delta: Q \times \Sigma \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma)$ 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>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>{ (q_2, 0) }</td>
<td>(q_3, \varepsilon)</td>
<td>(q_3, \varepsilon)</td>
<td>(q_3, \varepsilon)</td>
<td>(q_4, \varepsilon)</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>(q_3, \varepsilon)</td>
<td>{ (q_3, \varepsilon) }</td>
<td>(q_3, \varepsilon)</td>
<td>(q_3, \varepsilon)</td>
<td>(q_4, \varepsilon)</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>{ (q_2, $) }</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>{ (q_2, $) }</td>
<td>{ (q_3, \varepsilon) }</td>
<td>{ (q_3, \varepsilon) }</td>
<td>{ (q_4, \varepsilon) }</td>
<td>{ (q_4, \varepsilon) }</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.
\[ Q = \{ q_1, q_2, q_3, q_4 \}, \]
\[ \Sigma = \{ 0,1 \}, \]
\[ \Gamma = \{ 0, \$ \}, \]
\[ F = \{ q_1, q_4 \}, \text{ and} \]

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

| Input: | Stack: | | Stack: | | Stack: |
|--------|--------| |--------| |--------|
| 0, 1, \( \varepsilon \) | 0, \$ | \( \varepsilon \) | 0, \$ | \( \varepsilon \) | \( \varepsilon \) |
| \( q_1 \) | \( (q_2, 0) \) | 2 | \( (q_3, \varepsilon) \) | 3 | \( (q_4, \varepsilon) \) | 5 |
| \( q_2 \) | \( \{(q_2, \$)\} \) | | | | | |
| \( q_3 \) | | | | | | |
| \( q_4 \) | | | | | | |

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. \( \delta: Q \times \Sigma_e \times \Gamma_e \rightarrow \mathcal{P}(Q \times \Gamma_e) \) 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>( \epsilon )</th>
<th>1</th>
<th>( \epsilon )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack:</td>
<td>0</td>
<td>$</td>
<td>( \epsilon )</td>
<td>0</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>{ (q_2, 0) }</td>
<td>{ (q_3, \epsilon) }</td>
<td>( 1 )</td>
<td>{ (q_3, \epsilon) }</td>
</tr>
</tbody>
</table>

A pushdown automaton is a 6-tuple \((Q, \Sigma, \Gamma, \delta, q_0, F)\), where \(Q, \Sigma, \Gamma, \text{ and } F\) 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.
Last Time:

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

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

PDA \( M_3 \) recognizing the language \( \{ w w^R | w \in \{ 0, 1 \}^* \} \)
DFA Computation Rules

**Informally**

Given
- A DFA (~ a “Program”)
- and Input = string of chars, e.g. “1101”

A DFA computation (~ “Program run”):
- Start in start state

**Formally (i.e., mathematically)**

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

\[ w = w_1 w_2 \cdots w_n \]

A DFA computation is a sequence of states:

- specified by \( \hat{\delta}(q_0, w) \) where:
  \[ M \text{ accepts } w \text{ if } \hat{\delta}(q_0, w) \in F \]
  \[ M \text{ rejects } w \text{ otherwise} \]
DFA Multi-step Transition Function

\[ \hat{\delta} : Q \times \Sigma^* \rightarrow Q \]

- **Domain** (inputs):
  - state \( q \in Q \)
  - string \( w = w_1 w_2 \cdots w_n \) where \( w_i \in \Sigma \)
- **Range** (output):
  - state \( q \in Q \)

(Defined recursively)

Base case \[ \hat{\delta}(q, \varepsilon) = q \]

Recursive Case \[ \hat{\delta}(q, w'w_n) = \delta(\hat{\delta}(q, w'), w_n) \]
where \( w' = w_1 \cdots w_{n-1} \)

\( \delta : Q \times \Sigma \rightarrow Q \) is the transition function

A DFA computation is a sequence of states:
PDA Computation?

- **PDA = NFA + a stack**
  - Infinite memory
  - Push/pop top location only

A DFA computation is a sequence of states ...

A PDA computation is not just a sequence of states ...

... because the stack contents can change too!
PDA Configurations (IDs)

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

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

A sequence of configurations represents a PDA 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  Less 1 char  Push

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

\( q_1, q_2 \in Q \)

\( a \in \Sigma \)

\( w \in \Sigma^* \)

\( X \in \Gamma \)

\( \beta, \alpha \in \Gamma^* \)

**Multi-step**

- **Base Case** 0 steps

\[ I \vdash^* I \text{ for any ID } I \]

- **Recursive Case** > 0 steps

\[ I \vdash^* J \text{ if there exists some ID } K \]

such that \( I \vdash K \) and \( K \vdash^* J \)

Single step  Recursive “call”

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

\( q = \) the current state

\( w = \) the remaining input string

\( \gamma = \) the stack contents

This specifies the sequence of configurations for a PDA computation
PDA Running Input String Example

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

\[(q_1, 0011, \varepsilon) \vdash (q_2, 0011, \$)\]
\[\vdash (q_2, 011, 0\$)\]

- State
- Remaining Input
- Stack

Read 0, push 0

Input Read | Pop | Push
---|---|---
\[\varepsilon, \varepsilon \rightarrow \$\]
\[0, \varepsilon \rightarrow 0\]
\[1, 0 \rightarrow \varepsilon\]
\[\varepsilon, \$, \rightarrow \varepsilon\]
\[1, 0 \rightarrow \varepsilon\]
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

$\vdash (q_1, 0011, \varepsilon) \vdash (q_2, 0011, \$)$

$\vdash (q_2, 011, 0\$)$

$\vdash (q_2, 11, 00\$)$

$\vdash (q_3, 1, 0\$)$

Read 1, pop 0
PDA Running Input String Example

\[(q_1, 0011, \varepsilon) \Rightarrow (q_2, 0011, \$)\]
\[\Rightarrow (q_2, 011, 0\$)\]
\[\Rightarrow (q_2, 11, 00\$)\]
\[\Rightarrow (q_3, 1, 0\$)\]
\[\Rightarrow (q_3, \varepsilon, \$)\]
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., A DFA $M$ **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
PDAs and CFLs?

- **PDA** = NFA + a stack
  - Infinite memory
  - Push/pop top location only

- **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
  • We know: A CFL has a CFG describing it (definition of CFL)
  • To prove this part: show the CFG has an equivalent PDA

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

- $q \xrightarrow{xyz, \varepsilon \rightarrow \varepsilon} q$
- $q \xrightarrow{x, \varepsilon \rightarrow \varepsilon} q_1$
- $q_1 \xrightarrow{y, \varepsilon \rightarrow \varepsilon} q_2$
- $q_2 \xrightarrow{z, \varepsilon \rightarrow \varepsilon} q$

- Read $1$
- Read multi char

Read $1$
Shorthand: Multi-Stack Push Transition

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

- **Construct PDA from CFG** such that:
  - PDA accepts input only if CFG generates it
- **PDA:**
  - simulates generating a string with CFG rules
  - by (nondeterministically) trying all rules to find the right ones

![Diagram](image)
**CFG→PDA (sketch)**

- Construct PDA from CFG such that:
  - PDA accepts input only if CFG generates it
- PDA:
  - simulates generating a string with CFG rules
  - by (nondeterministically) trying all rules to find the right ones

![Diagram of PDA states]

- **$q_{start}$**
  - Push start variable onto stack
  - If: stack top is variable $A$, pop and ...  
  - ... push rule's right-sides (nondeterministically)

- **$q_{loop}$**
  - $\varepsilon, S \rightarrow \varepsilon$ for rule $A \rightarrow w$
  - $a, a \rightarrow \varepsilon$ for terminal $a$
  - If: stack top is terminal $a$, pop and ...
  - ... read matching input

- **$q_{accept}$**
Example **CFG→PDA**

- **S → aTb | b**
- **T → Ta | ε**

- **push start variable onto stack**

- **If: stack top is variable S, pop S and ...**

- **push rule right-sides (in rev order)**
Example **CFG → PDA**

- **Start State:** $q_{\text{start}}$
  - $\epsilon, \epsilon \rightarrow S$
  - $\epsilon, \epsilon \rightarrow \$ $

- **Loop State:** $q_{\text{loop}}$
  - $\epsilon, S \rightarrow b$
  - $\epsilon, \epsilon \rightarrow T$
  - $\epsilon, \epsilon \rightarrow a$
  - $\epsilon, T \rightarrow a$
  - $\epsilon, \epsilon \rightarrow T$
  - $\epsilon, S \rightarrow b$
  - $\epsilon, T \rightarrow \epsilon$
  - $a, a \rightarrow \epsilon$
  - $b, b \rightarrow \epsilon$

**Grammar Rules:**

- $S \rightarrow aTb \mid b$
- $T \rightarrow Ta \mid \epsilon$
Example **CFG→PDA**

### Grammar

- **S → aTb | b**
- **T → Ta | ε**

### PDA

- **q_{start}**
- **q_{loop}**
- **q_{accept}**

- **ε,ε→$**
- **ε,ε→S**
- **ε,S→b**
- **ε,T→a**
- **ε,S→b**
- **ε,T→ε**
- **a,a→ε**
- **b,b→ε**

- **ε,ε→T**
- **ε,ε→a**
- **ε,ε→T**

---

If: stack top is **terminal**, **pop** and read matching input.
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 \( T \rightarrow \varepsilon \))

Machine is doing reverse of grammar:
- start with the string,
- Find rules that generate string

PDA Example

<table>
<thead>
<tr>
<th>State</th>
<th>Input</th>
<th>Stack</th>
<th>Equiv Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>(q_{\text{start}})</td>
<td>aab</td>
<td>(\varepsilon)</td>
<td>$S \rightarrow aTb$</td>
</tr>
<tr>
<td>(q_{\text{loop}})</td>
<td>aab</td>
<td>(S$)</td>
<td></td>
</tr>
<tr>
<td>(q_{\text{loop}})</td>
<td>aab</td>
<td>(aTb$)</td>
<td>$S \rightarrow aTb$</td>
</tr>
<tr>
<td>(q_{\text{loop}})</td>
<td>ab</td>
<td>(Tb$)</td>
<td></td>
</tr>
<tr>
<td>(q_{\text{loop}})</td>
<td>ab</td>
<td>(Tab$)</td>
<td>$T \rightarrow Ta$</td>
</tr>
<tr>
<td>(q_{\text{loop}})</td>
<td>ab</td>
<td>(ab$)</td>
<td>$T \rightarrow \varepsilon$</td>
</tr>
<tr>
<td>(q_{\text{loop}})</td>
<td>b</td>
<td>(b$)</td>
<td></td>
</tr>
<tr>
<td>(q_{\text{loop}})</td>
<td></td>
<td>($)</td>
<td></td>
</tr>
<tr>
<td>(q_{\text{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 \( T \rightarrow \varepsilon \))

If: stack top is variable \( S \), **pop S** and **push** rule right-sides (in rev order)

<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>( S$ )</td>
<td>( S \rightarrow aTb )</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>( 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>( T \rightarrow \varepsilon )</td>
</tr>
<tr>
<td>( q_{accept} )</td>
<td>$</td>
<td>$</td>
<td>$</td>
</tr>
</tbody>
</table>

PDA Example
**Example CFG → PDA**

**Example Derivation using CFG:**

1. $S \Rightarrow aTb$ (using rule $S \rightarrow aTb$)
2. $\Rightarrow aTab$ (using rule $T \rightarrow Ta$)
3. $\Rightarrow aab$ (using rule $T \rightarrow \varepsilon$)

**CFG:**

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

**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>b$S$</td>
<td>$S \rightarrow aTb$</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>$7b$</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$\varepsilon$</td>
<td></td>
</tr>
<tr>
<td>$q_{accept}$</td>
<td>$\varepsilon$</td>
<td>$\varepsilon$</td>
<td></td>
</tr>
</tbody>
</table>

If stack top is terminal, pop and read matching input.
Example $\text{CFG} \rightarrow \text{PDA}$

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>$S \rightarrow aTb$</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>
A lang is a CFL iff some PDA recognizes it

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

⇐ If a PDA recognizes a language, then it’s a CFL
  • To prove this part: show PDA has an equivalent CFG
PDA→CFG: Prelims

Before converting PDA to CFG, modify it so:

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

\textbf{Important:}
This doesn't change the language recognized by the PDA
PDA $P \rightarrow$ CFG $G$: Variables

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

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 $\{A_{pq} \mid p, q \in Q\}$

• **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 $P$ -> 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

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

$\Leftarrow$ If a PDA recognizes a language, then it's a CFL
  • Convert PDA to CFG
Submit in-class work 3/20

On gradescope