UMB CS 420

Pushdown Automata (PDAs)

Wednesday, February 23, 2022
Announcements

• HW 3 in

• HW 4 out
  • Due Sun 2/27 11:59pm EST
Last Time: Generating Strings with a CFG

A CFG represents a context free language!

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

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

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

**KEY DIFFERENCE:**

<table>
<thead>
<tr>
<th>A Regular lang is <strong>defined</strong> with a FSM</th>
<th>A CFL is <strong>defined</strong> 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)

PDA = NFA + 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

when machine starts:
- don’t read input,
- don’t pop anything,
- push empty stack symbol

accept only when stack is empty

Read input → Pop → Push

$ = \text{special symbol, indicating empty stack}$

read 0, no pop, push 0 (and repeat)

read 1, pop 0, no push (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. \(\delta : Q \times \Sigma \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma)\) is the transition function,
5. \(q_0 \in Q\) is the start state, and
6. \(F \subseteq Q\) is the set of accept states.

Non-deterministic: produces a set of \((\text{State}, \text{Stack Char})\) pairs

Stack alphabet can have special stack symbols, e.g., $
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. \(\delta: Q \times \Sigma \times \Gamma \rightarrow P(Q \times \Gamma)\) 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: $</th>
<th>\epsilon</th>
<th>0</th>
<th>1</th>
<th>\epsilon</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$ \epsilon</td>
<td>0</td>
<td>$</td>
<td>\epsilon</td>
<td></td>
</tr>
<tr>
<td>\epsilon</td>
<td>0</td>
<td>$ \epsilon</td>
<td>0</td>
<td>$</td>
<td>\epsilon</td>
</tr>
</tbody>
</table>

The transition function \( \delta \) is defined as follows:

- \( \delta(q_1, 0) = \{ (q_2, 0) \} \)
- \( \delta(q_2, \epsilon) = \{ (q_3, \epsilon) \} \)
- \( \delta(q_3, 0) = \{ (q_4, \epsilon) \} \)
- \( \delta(q_4, \$) = \{ (q_2, \$) \} \)

A pushdown automaton is a 6-tuple \( (Q, \Sigma, \Gamma, \delta, q_0, F) \), where \( Q, \Sigma, \Gamma, \delta, \) 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.
\[
Q = \{q_1, q_2, q_3, q_4\}, \\
\Sigma = \{0,1\}, \\
\Gamma = \{0,\$$\}, \\
F = \{q_1, q_4\}, \text{ and}
\]

\(\delta\) 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_3, \varepsilon)}</td>
<td>2</td>
</tr>
<tr>
<td>(q_2)</td>
<td>{(q_3, \varepsilon)}</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>(q_3)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(q_4)</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. \(\delta: Q \times \Sigma_e \times \Gamma_e \rightarrow \mathcal{P}(Q \times \Gamma_e)\) is the transition function,
5. \(q_0 \in Q\) is the start state, and
6. \(F \subseteq Q\) is the set of accept states.
\[ Q = \{ q_1, q_2, q_3, q_4 \}, \]
\[ \Sigma = \{0, 1\}, \]
\[ \Gamma = \{0, \$\}, \]
\[ F = \{ q_1, q_4 \}, \]

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

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

A \textit{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. \( \delta: Q \times I \Sigma \times I \Gamma \rightarrow P(Q \times I \Gamma) \) 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>Stack:</th>
<th>Input:</th>
<th>0</th>
<th>1</th>
<th>( \epsilon )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 $ \epsilon \</td>
<td>0 $ \epsilon \</td>
<td>0 $ \epsilon \</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{array}{c|c|c|c|c}
q_1 & q_2 & q_3 & q_4 \\
\{(q_2, 0)\} & \{(q_3, \epsilon)\} & \{(q_3, \epsilon)\} & \{(q_4, \epsilon)\} \\
1 & 2 & 3 & 4 & 5 \\
\end{array}
\]

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>(1)</th>
<th>$\varepsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack:</td>
<td>$0$ $\varepsilon$</td>
<td>$0$ $\varepsilon$</td>
<td>$\varepsilon$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>${(q_2, 0)}$</td>
<td>${(q_3, \varepsilon)}$</td>
<td>${(q_4, \varepsilon)}$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$1$</td>
<td>$2$</td>
<td>$4$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$3$</td>
<td>$5$</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.
Flashback: DFA Computation Model

**Informally**

- **Computer** = a DFA

- **Program** = input string of chars, e.g. “1101”

To run a program:

- **Start** in “start state”

- **Read** 1 char at a time, changing states according to the transition table

- **Result** =
  - “Accept” if last state is “Accept” state
  - “Reject” otherwise

---

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

- \( M = (Q, \Sigma, \delta, q_0, F) \)

- \( w = w_1 w_2 \cdots w_n \)

- \( r_0 = q_0 \)

- \( \delta(r_i, w_{i+1}) = r_{i+1}, \text{ for } i = 0, \ldots, n - 1 \)

Sequence of states completely represents a computation

- \( M \) accepts \( w \) if
  - sequence of states \( r_0, r_1, \ldots, r_n \) in \( Q \) exists . . .
  - with \( r_n \in F \)
PDA Configurations (IDs)

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

• A configuration (or ID) \((q, w, \gamma)\) has three 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

Single-step

Before / After configurations

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

- 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
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
PDA Running Input String Example

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

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

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

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

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

\[\vdash (q_3, 1, 0\$)\]
PDA Running Input String Example

\[(q_1, 0011, \varepsilon) \xrightarrow{\varepsilon, \varepsilon \rightarrow \$} (q_2, 0011, \$)\]
\[(q_2, 011, 0\$)\]
\[(q_2, 11, 00\$)\]
\[(q_3, 1, 0\$)\]
\[(q_3, \varepsilon, \$)\]
PDA Running Input String Example

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

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

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

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

\[\vdash (q_3, \varepsilon, \$)\]

\[\vdash (q_4, \varepsilon, \varepsilon)\]
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

```
q_{start} → q_{loop} → q_{accept}

ε, ε → S$

ε, A → w  for rule A → w
a, a → ε   for terminal a
ε, $ → ε  
```
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
Example **CFG→PDA**

Transition rules:
- \( S \rightarrow aTb \mid b \)
- \( T \rightarrow Ta \mid \varepsilon \)

- \( \varepsilon, S \rightarrow b \rightarrow \varepsilon \)
- \( \varepsilon, T \rightarrow a \rightarrow \varepsilon \)
- \( \varepsilon, T \rightarrow \varepsilon \rightarrow \varepsilon \)
- \( a, a \rightarrow \varepsilon \)
- \( b, b \rightarrow \varepsilon \)

Transition actions:
- If stack top is **variable** \( S \), pop \( S \) and push rule right-sides (in rev order)
Example \textbf{CFG$\rightarrow$PDA}

\begin{align*}
S & \rightarrow aTb \mid b \\
T & \rightarrow Ta \mid \varepsilon
\end{align*}
Example **CFG→PDA**

$$S \rightarrow aTb \mid b$$

$$T \rightarrow Ta \mid \varepsilon$$

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

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

- **$q_{accept}$**
  - $\varepsilon, S \rightarrow b$
  - $\varepsilon, T \rightarrow \varepsilon$
  - $a, a \rightarrow \varepsilon$
  - $b, b \rightarrow \varepsilon$

*If stack top is a terminal, pop and read matching input.*
Example CFG→PDA

S → aTb | b
T → Ta | ε

Example Derivation using CFG:
S ⇒ aTb (using rule S → aTb)
⇒ aTab (using rule T → Ta)
⇒ aab (using rule T → ε)

PDA Example

<table>
<thead>
<tr>
<th>State</th>
<th>Input</th>
<th>Stack</th>
<th>Equiv Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>qstart</td>
<td>aab</td>
<td></td>
<td></td>
</tr>
<tr>
<td>qloop</td>
<td>aab</td>
<td>$S$</td>
<td>S → aTb</td>
</tr>
<tr>
<td>qloop</td>
<td>aab</td>
<td>aTb$</td>
<td></td>
</tr>
<tr>
<td>qloop</td>
<td>ab</td>
<td>Tb$</td>
<td></td>
</tr>
<tr>
<td>qloop</td>
<td>ab</td>
<td>Tab$</td>
<td>T → Ta</td>
</tr>
<tr>
<td>qloop</td>
<td>ab</td>
<td>ab$</td>
<td>T → ε</td>
</tr>
<tr>
<td>qloop</td>
<td>b</td>
<td>b$</td>
<td></td>
</tr>
<tr>
<td>qaccept</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example \textbf{CFG}→\textbf{PDA}

**Example Derivation using CFG:**

\[ S \Rightarrow aTb \text{ (using rule } S \Rightarrow aTb) \]
\[ \Rightarrow aTab \text{ (using rule } T \Rightarrow Ta) \]
\[ \Rightarrow aab \text{ (using rule } T \Rightarrow \varepsilon) \]

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>$q_{\text{start}}$</td>
<td>aab</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$q_{\text{loop}}$</td>
<td>aab</td>
<td></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{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$</td>
<td>( S \rightarrow aTb )</td>
</tr>
<tr>
<td>( q_{loop} )</td>
<td>ab</td>
<td>7b$</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_{loop} )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( q_{accept} )</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

if stack top is a 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 \( T \rightarrow \varepsilon \))

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>( 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>( Tab$ )</td>
<td>( T \rightarrow Ta )</td>
</tr>
<tr>
<td>( q_{loop} )</td>
<td>ab</td>
<td>ab$ $</td>
<td>( T \rightarrow \varepsilon )</td>
</tr>
<tr>
<td>( q_{loop} )</td>
<td>b</td>
<td>b$ $</td>
<td>( T \rightarrow \varepsilon )</td>
</tr>
<tr>
<td>( q_{accept} )</td>
<td></td>
<td>$</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, $q_{accept}$.
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

$P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{\text{accept}}\})$  \hspace{1cm} \text{variables of } G \text{ 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 $P \rightarrow\text{ CFG } G$ : 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 $G$
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 $G$
A language is a CFL ⇔ A 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
  • Convert PDA⇒CFG
Check-in Quiz 2/23

On Gradescope