Pushdown Automata (PDAs)

Monday, October 7, 2020
HW3 Questions?
HW4 out

• HW4 due in 2 weeks

• HW3 due Sunday 11:59pm EST
Last time: Designing Grammars

• Start with small grammars and then combine (just like FSMs)

• “Or”: \[ S \rightarrow S_1 \mid S_2 \]

• “Concatenate”: \[ S \rightarrow S_1 S_2 \]

• “Repetition”: \[ S' \rightarrow S' S_1 \mid \epsilon \]
In-class exercise: Designing grammars

alphabet $\Sigma$ is $\{0, 1\}$

$\{w \mid w$ starts and ends with the same symbol$\}$

- $S \rightarrow 0C'0 \mid 1C'1 \mid \varepsilon$  “string starts/ends with same symbol, middle can be anything”
- $C' \rightarrow C'C \mid \varepsilon$  “all possible terminals, repeated (ie, all possible strings)”
- $C \rightarrow 0 \mid 1$  “all possible terminals”
## Analogies

<table>
<thead>
<tr>
<th>Regular Language</th>
<th>Context-Free Language (CFL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular Expression (Regexp)</td>
<td>Context-Free Grammar (CFG)</td>
</tr>
<tr>
<td>Regexp describes a Reg lang</td>
<td>CFG describes/generates a CFL</td>
</tr>
</tbody>
</table>

39
# Analogies

<table>
<thead>
<tr>
<th>Regular Language</th>
<th>Context-Free Language (CFL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular Expression (Regexp)</td>
<td>Context-Free Grammar (CFG)</td>
</tr>
<tr>
<td>Regexp describes a Reg lang</td>
<td>CFG describes/generates a CFL</td>
</tr>
<tr>
<td><strong>TODAY:</strong></td>
<td></td>
</tr>
<tr>
<td>Finite automaton (FSM)</td>
<td>Push-down automaton (PDA)</td>
</tr>
<tr>
<td>FSM recognizes a regular lang</td>
<td>PDA recognizes a CFL</td>
</tr>
</tbody>
</table>

TODAY:

- Context-Free Grammar (CFG)
- CFG describes/generates a CFL
# Analogies

<table>
<thead>
<tr>
<th>Regular Language</th>
<th>Context-Free Language (CFL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular Expression (Regexp)</td>
<td>Context-Free Grammar (CFG)</td>
</tr>
<tr>
<td>Regexp describes a Reg lang</td>
<td>CFG describes/generates a CFL</td>
</tr>
<tr>
<td>Finite automaton (FSM)</td>
<td>Push-down automaton (PDA)</td>
</tr>
<tr>
<td>FSM recognizes a regular lang</td>
<td>PDA recognizes a CFL</td>
</tr>
<tr>
<td><strong>DIFFERENCE:</strong></td>
<td><strong>DIFFERENCE:</strong></td>
</tr>
<tr>
<td>Regular lang defined via FSM</td>
<td>CFL defined via CFG</td>
</tr>
<tr>
<td>Must prove Regexp $\Leftrightarrow$ Reg lang</td>
<td>Must prove PDA $\Leftrightarrow$ CFL</td>
</tr>
</tbody>
</table>
Pushdown Automata (PDA)

- PDA = NFA + a stack
A (Mathematical) Stack Specification

- Access to top element of stack only
- Operations: push, pop

- (What could be a possible code representation?)
Pushdown Automata (PDA)

• PDA = NFA + a stack
  • Infinite memory
  • But only read/write top loc
    • Push/pop
A Example PDA

\{0^n1^n | n \geq 0\}

Read input

\(\varepsilon, \varepsilon \rightarrow \$\)

Pop

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

0, \varepsilon \rightarrow 0

“read 0, push 0 (repeat)”

1, 0 \rightarrow \varepsilon

“read 1, pop 0 (repeat)”

\varepsilon, \$, \varepsilon \rightarrow \varepsilon

“when machine starts, don’t read input, don’t pop, and push empty stack symbol”

\varepsilon, \$, \varepsilon \rightarrow \varepsilon

“accept when stack is empty”
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.
A pushdown automaton is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where $Q$, $\Sigma$, and $\Gamma$ 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>Stack:</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>( \varepsilon )</td>
<td>1</td>
<td>( \varepsilon )</td>
</tr>
<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>
</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.

\( \varepsilon, \$ \rightarrow \) $
\[ Q = \{ q_1, q_2, q_3, q_4 \}, \]
\[ \Sigma = \{0,1\}, \]
\[ \Gamma = \{0, $\}, \]
\[ F = \{ q_1, q_4 \}, \] and

\( \delta \) is given by the following table, wherein blank entries signify 0.

<table>
<thead>
<tr>
<th>Stack:</th>
<th>Input:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ \mid \varepsilon \mid 0 \mid \varepsilon \mid 0 \mid \varepsilon \mid \varepsilon \mid \varepsilon</td>
<td></td>
</tr>
<tr>
<td>\varepsilon \mid { (q_2, 0) } \mid { (q_3, \varepsilon) } \mid 2 \mid { (q_3, \varepsilon) } \mid 3 \mid { (q_4, \varepsilon) } \mid 5</td>
<td></td>
</tr>
<tr>
<td>q_1 \mid q_2 \mid q_3 \mid q_4</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>0 $ ( \varepsilon )</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>2</td>
<td>{(q_2, $)}</td>
</tr>
<tr>
<td>( q_2 )</td>
<td></td>
<td>{(q_3, \varepsilon)}</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>( q_3 )</td>
<td>{(q_4, \varepsilon)}</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( q_4 )</td>
<td></td>
<td></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.
\[ Q = \{ q_1, q_2, q_3, q_4 \}, \]
\[ \Sigma = \{ 0, 1 \}, \]
\[ \Gamma = \{ 0, \$ \}, \]
\[ F = \{ q_1, q_4 \}, \]

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

<table>
<thead>
<tr>
<th>Input:</th>
<th>0</th>
<th>( \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>
</tbody>
</table>

\[
\begin{align*}
q_1 & \quad \{(q_2, 0)\} \\
q_2 & \quad \{(q_3, \varepsilon)\} \\
q_3 & \quad \{(q_3, \varepsilon)\} \\
q_4 & \quad \{(q_4, \varepsilon)\}
\end{align*}
\]

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>Stack:</th>
<th>0</th>
<th>( $ )</th>
<th>( \varepsilon )</th>
<th>1</th>
<th>( $ )</th>
<th>( \varepsilon )</th>
<th>0</th>
<th>( $ )</th>
<th>( \varepsilon )</th>
<th>( \varepsilon )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input:</td>
<td>0</td>
<td>( \varepsilon )</td>
<td>( (q_2, 0) )</td>
<td>( (q_3, \varepsilon) )</td>
<td>2</td>
<td>( (q_3, \varepsilon) )</td>
<td>3</td>
<td>( (q_4, \varepsilon) )</td>
<td>4</td>
<td>( (q_2, $) )</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. \( \delta: Q \times \Sigma \varepsilon \times \Gamma \varepsilon \rightarrow \mathcal{P}(Q \times \Gamma \varepsilon) \) is the transition function,
5. \( q_0 \in Q \) is the start state, and
6. \( F \subseteq Q \) is the set of accept states.
Pushdown Automata (PDA)

- PDA = NFA + a stack
  - Infinite memory
  - But only read/write top location
    - Push/pop

- Want to prove: PDA $\Leftrightarrow$ CFG

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

• => If CFL, then PDA recognizes it
  • (Easier)
    • All CFLs have CFG describing it (definition of CFL)
    • Convert CFG -> PDA

• <= If PDA recognizes, then CFL
CFG -> PDA

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

- Intuitively, PDA nondeterministically tries all rules

```
q_{start} → q_{loop}

ε, ε → S$

q_{loop} → q_{accept}

ε, $ → ε

ε, A → w for rule A → w

a, a → ε for terminal a

“push start variable onto stack”

“if variable on stack top, pop and nondet. replace with every right-side”

“if terminal on stack top, read input and pop”
```
Example CFG -> PDA

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

"pop S and replace with rule right-side"

- \( \varepsilon, S \rightarrow b \)
- \( \varepsilon, \varepsilon \rightarrow T \)
- \( \varepsilon, \varepsilon \rightarrow a \)
- \( \varepsilon, T \rightarrow a \)
- \( \varepsilon, \varepsilon \rightarrow T \)
- \( \varepsilon, S \rightarrow b \)
- \( \varepsilon, T \rightarrow \varepsilon \)
- \( a, a \rightarrow \varepsilon \)
- \( b, b \rightarrow \varepsilon \)
Example CFG -> PDA

$S \rightarrow aTb \mid b$

$T \rightarrow Ta \mid \varepsilon$

$q_{start}$

$\varepsilon, \varepsilon \rightarrow \$ 

$\varepsilon, \varepsilon \rightarrow S$

$\varepsilon, S \rightarrow b$

$\varepsilon, T \rightarrow a$

$q_{loop}$

$\varepsilon, T \rightarrow \varepsilon$

$\varepsilon, \varepsilon \rightarrow \varepsilon$

$q_{accept}$

$\varepsilon, \varepsilon \rightarrow a$

$\varepsilon, \varepsilon \rightarrow T$

$\varepsilon, \varepsilon \rightarrow T$

$a, a \rightarrow \varepsilon$

$b, b \rightarrow \varepsilon$
Example CFG -> PDA

$S \rightarrow aTb \mid b$

$T \rightarrow Ta \mid \epsilon$

```
q_{start}
```

$
\epsilon, \epsilon \rightarrow S$

$
\epsilon, \epsilon \rightarrow T$

$
\epsilon, \epsilon \rightarrow a$

```
q_{loop}
```

```
q_{accept}
```

```
\epsilon, S \rightarrow b
\epsilon, T \rightarrow a
\epsilon, \epsilon \rightarrow T
\epsilon, \epsilon \rightarrow T
\epsilon, S \rightarrow b
\epsilon, T \rightarrow \epsilon
a, a \rightarrow \epsilon
b, b \rightarrow \epsilon
```

“if terminal on stack top, read input and pop”
A lang is a CFL iff some PDA recognizes it

• => If CFL, then PDA recognizes it
  • (Easier)
  • All CFLs have CFG describing it (definition of CFL)
  • Convert CFG -> PDA (DONE!)
• <= If PDA recognizes, then CFL
  • (Harder)
  • Need PDA -> CFG
PDA -> CFG: Step 1

Before converting PDA to CFG, modify it (without changing its lang) 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.

(confirm this to yourselves)
PDA \( P \rightarrow \) CFG \( G \): substitution rules

• For every pair of states \( p, q \): add grammar variable \( A_{pq} \)

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

• \( A_{pq} \) represents “all possible inputs read going from state \( p \) to \( q \)”

• Add rules: \( A_{pq} \rightarrow A_{pr}A_{rq} \), for every state \( r \)
  • “All possible strings when going from \( p \) to \( q \) =
    • All possible strings going from \( p \) to \( r \), concatenated with
    • All possible strings going from \( r \) to \( q \)”

• We still need rules that produce terminals!

• The key: pair up stack pushes and pops (essence of CFL)
PDA $P \rightarrow$ CFG $G$: generating strings

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

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

- The key: pair up stack pushes and pops (essence of 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 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 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 lang is a CFL ⇔ A PDA recognizes it

• \( \Rightarrow \) If CFL, then PDA recognizes it
  • All CFLs have CFG describing it (definition of CFL)
  • Convert CFG \( \rightarrow \) PDA

• \( \Leftarrow \) If PDA recognizes, then CFL
  • Convert PDA \( \rightarrow \) CFG
Regular languages are CFLs, prove 3 ways

- NFA -> PDA (with no stack moves) -> CFG
  - Just now
- DFA -> CFG
  - Textbook page 107
- Regexp -> CFG
  - HW4
Check-in Quiz 10/7
On Gradescope
End of Class Survey 10/7
See course website

Remember, no class next Monday!