Pushdown Automata (PDAs)

Wednesday, March 3, 2021
Announcements

• HW5 deadline extended
  • Now due: Wed 3/10 11:59pm EST

• Reminder: Spring Break Mon 3/15 – Sun 3/21
  • No classes
Last Time:

<table>
<thead>
<tr>
<th>Regular Languages</th>
<th>Context-Free Languages (CFLs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular Expression (Regexp)</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>Regular Languages</th>
<th>Context-Free Languages (CFLs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular Expression (Regexp)</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>
<tr>
<td>Finite automaton (FSM)</td>
<td><strong>TODAY:</strong></td>
</tr>
<tr>
<td>An FSM <strong>recognizes</strong> a Regular lang</td>
<td>Push-down automaton (PDA)</td>
</tr>
<tr>
<td></td>
<td>A PDA <strong>recognizes</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 (Regexp)</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>
<tr>
<td>Finite automaton (FSM)</td>
<td>Push-down automaton (PDA)</td>
</tr>
<tr>
<td>An FSM recognizes a Regular lang</td>
<td>A PDA recognizes a CFL</td>
</tr>
<tr>
<td><strong>DIFFERENCE:</strong></td>
<td><strong>DIFFERENCE:</strong></td>
</tr>
<tr>
<td>A Regular lang is defined with a FSM</td>
<td>A CFL is defined with a CFG</td>
</tr>
<tr>
<td><em>Must prove:</em> Reg expr ⇔ Reg lang</td>
<td><em>Must prove:</em> PDA ⇔ 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 data representation in code?)
Pushdown Automata (PDA)

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

```
NFA states
```

```
<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
</table>
```

- Input: \(a\ a\ a\ b\ b\)
- Stack: \(|x y z|\)

48
An Example PDA

\[\{0^n1^n \mid n \geq 0\}\]

- Read input
- Pop
- Push

q1

- \(\epsilon, \epsilon \rightarrow \$\)
- when machine starts:
  - don’t read input,
  - don’t pop anything,
  - push empty stack symbol

q2

- \(0, \epsilon \rightarrow 0\)
- \((\text{nondeterministically})\)
  - read 1, pop 0, no push (and repeat)

q3

- \(1, 0 \rightarrow \epsilon\)
- accept only when stack is empty

q4

- \(\epsilon, \$, \epsilon\)
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, Stack alphabet can have special stack symbols, e.g., \$
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.
In-class example

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

\( \delta \) 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>{ (q_2, 0) }</td>
<td>{ (q_3, \varepsilon) }</td>
<td>2</td>
<td>{ (q_3, \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. \(\delta: Q \times \Sigma \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma)\) 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>
</tr>
</thead>
<tbody>
<tr>
<td>Stack:</td>
<td>$</td>
<td>\varepsilon</td>
<td>0</td>
</tr>
</tbody>
</table>

| \( q_1 \) | \{ (q_2, 0) \} |
| \( q_2 \) | \{ (q_3, \varepsilon) \} |
| \( q_3 \) | \{ (q_3, \varepsilon) \} |
| \( q_4 \) | \{ (q_4, \varepsilon) \} |

A pushdown automaton is a 6-tuple \((Q, \Sigma, \Gamma, \delta, q_0, F)\), where \(Q, \Sigma, \Gamma, F\) 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>( \epsilon )</th>
<th>( 1 )</th>
<th>( \epsilon )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack:</td>
<td>0 ( $ ) ( \epsilon )</td>
<td>0 ( $ ) ( \epsilon )</td>
<td>( \epsilon )</td>
<td>( \epsilon )</td>
</tr>
<tr>
<td>0 ( \epsilon )</td>
<td>( {q_2, 0} )</td>
<td>( {q_3, \epsilon} )</td>
<td>2</td>
<td>( {q_4, \epsilon} )</td>
</tr>
<tr>
<td>1 ( \epsilon )</td>
<td>( {q_3, \epsilon} )</td>
<td>3</td>
<td>( {q_4, \epsilon} )</td>
<td></td>
</tr>
</tbody>
</table>

A *pushdown automaton* is a 6-tuple \((Q, \Sigma, \Gamma, \delta, q_0, F)\), where \( Q, \Sigma, \Gamma, \delta, q_0, F \) 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.
\[ Q = \{q_1, q_2, q_3, q_4\}, \]
\[ \Sigma = \{0, 1\}, \]
\[ \Gamma = \{0, \$\}, \]
\[ F = \{q_1, q_4\}, \text{ and} \]
\[ \delta \text{ is given by the following table, wherein blank entries signify } \emptyset. \]

<table>
<thead>
<tr>
<th>Stack:</th>
</tr>
</thead>
</table>
| \begin{tabular}{c|c|c|c|c|c}
| 0 & \$ & \(\varepsilon\) & 0 & \$ & \(\varepsilon\) & \(\varepsilon\) \\
| \hline
| \(q_1\) & \{(q_2, 0)\} & \{(q_3, \varepsilon)\} & 2 & \{(q_3, \varepsilon)\} & 3 & \{(q_4, \varepsilon)\} & 4 \\
| \hline
| \(q_2\) & \(1\) & \(1\) & \(1\) & \(1\) & \(1\) & \(1\) & \(1\) \\
| \hline
| \(q_3\) & \(0, \varepsilon \rightarrow \$\) & \(1, 0 \rightarrow \varepsilon\) & \(1, 0 \rightarrow \varepsilon\) & \(1, 0 \rightarrow \varepsilon\) & \(1, 0 \rightarrow \varepsilon\) & \(1, 0 \rightarrow \varepsilon\) & \(1, 0 \rightarrow \varepsilon\) \\
| \hline
| \(q_4\) & \(\varepsilon, \$ \rightarrow \varepsilon\) & \(\varepsilon, \$ \rightarrow \varepsilon\) & \(\varepsilon, \$ \rightarrow \varepsilon\) & \(\varepsilon, \$ \rightarrow \varepsilon\) & \(\varepsilon, \$ \rightarrow \varepsilon\) & \(\varepsilon, \$ \rightarrow \varepsilon\) & \(\varepsilon, \$ \rightarrow \varepsilon\) \\
| \hline
\end{tabular} |

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

\[ \delta \text{ is given by the following table, wherein blank entries signify } \emptyset. \]

<table>
<thead>
<tr>
<th>Input:</th>
<th>Stack:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 1,</td>
<td>$</td>
</tr>
<tr>
<td>Stack:</td>
<td>$</td>
</tr>
<tr>
<td>q_1</td>
<td>($q_2, 0) }$</td>
</tr>
<tr>
<td>q_2</td>
<td>($q_3, $) }$</td>
</tr>
<tr>
<td>q_3</td>
<td>{($q_2, $)}</td>
</tr>
<tr>
<td>q_4</td>
<td>{($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. \(\delta: Q \times \Sigma \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma)\) is the transition function,
5. \(q_0 \in Q\) is the start state, and
6. \(F \subseteq Q\) is the set of accept states.
Pushdown Automata (PDA)

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

- **Want to prove**: PDA $\Leftrightarrow$ CFG

- Then, to prove that a language is context-free, we can either:
  - Create a CFG, or
  - Create a PDA
CFL $\iff$ 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)
    • To prove forward dir: Convert CFG -> PDA
  • <= If a PDA recognizes a language, then it’s a CFL
CFG -> PDA

- 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} \rightarrow \epsilon, \epsilon \rightarrow S$

```

```
q_{loop}

```

```
q_{accept}

```

- push start variable onto stack

```
\epsilon, A \rightarrow w \quad \text{for rule } A \rightarrow w

```

```
a, a \rightarrow \epsilon \quad \text{for terminal } a

```
Transition with multiple stack pushes
CFG -> PDA

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

• Intuitively, PDA will **nondeterministically** try all rules

- $q_{start}$
  - $\epsilon, \epsilon \rightarrow S$:
    - Push start variable onto stack

- $q_{loop}$
  - $\epsilon, \epsilon \rightarrow \epsilon$:

- $q_{accept}$
  - $\epsilon, A \rightarrow w$ for rule $A \rightarrow w$
  - $a, a \rightarrow \epsilon$ for terminal $a$
  - If stack top is a **terminal**, pop and read matching input

- If stack top is a **variable**, pop and (nondet) push rule’s right-side
Example CFG -> PDA

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

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

\[ \varepsilon, S \rightarrow b \]
\[ \varepsilon, T \rightarrow a \]
\[ \varepsilon, \varepsilon \rightarrow T \]
\[ \varepsilon, \varepsilon \rightarrow a \]
\[ \varepsilon, \varepsilon \rightarrow \varepsilon \]
\[ a, a \rightarrow \varepsilon \]
\[ b, b \rightarrow \varepsilon \]
Example CFG -> PDA

\[
S \rightarrow aTb \mid b
\]
\[
T \rightarrow Ta \mid \varepsilon
\]
Example CFG -> PDA

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

$$\varepsilon, S \rightarrow b$$
$$\varepsilon, T \rightarrow a$$
$$\varepsilon, \varepsilon \rightarrow T$$

if stack top is a terminal, pop and read matching input
Example CFG -> PDA

Example Derivation using CFG:
\[ S \rightarrow aTb \mid b \]
\[ T \rightarrow Ta \mid \varepsilon \]
\[ aab \]

PDA Example, input aab

<table>
<thead>
<tr>
<th>Input read</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>aTb</td>
</tr>
<tr>
<td>a</td>
<td>Tb \rightarrow Tab \rightarrow ab</td>
</tr>
<tr>
<td>aa</td>
<td>b \rightarrow aab</td>
</tr>
<tr>
<td>aab</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
  • (Easier)
  • **We know:** A CFL has a CFG describing it (definition of CFL)
  • **Need to:** Convert CFG -> PDA **(DONE!)**

• <= If a PDA recognizes a language, then it’s a CFL
  • (Harder)
  • **Need to:** Convert PDA -> 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 $\text{push}$ move) or pops one off the stack (a $\text{pop}$ move), but it does not do both at the same time.

Important:
This doesn’t change the language recognized by the PDA (confirm this to yourselves)
PDA $P$ $\rightarrow$ CFG $G$: Variables

$P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{\text{accept}}\})$  \hspace{1cm} 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 \(^{(\text{essence of a CFL})^3}\)
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 $P \to \text{CFG } G$: 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 $\iff$ A PDA recognizes it

• $\Rightarrow$ If a language is a CFL, then a PDA recognizes it
  • We know: A CFL has a CFG describing it (definition of CFL)
  • Need to: Convert CFG $\rightarrow$ PDA (DONE!)

• $\Leftarrow$ If a PDA recognizes a language, then it’s a CFL
  • Need to: Convert PDA $\rightarrow$ CFG (DONE!)
Regular languages are CFLs: 3 Proofs

• NFA -> PDA (with no stack moves) -> CFG
  • Just now
• DFA -> CFG
  • Textbook page 107
• Regular expression -> CFG
  • HW5
Check-in Quiz 3/3

On Gradescope