Equivalence of CFGs and PDAs

A language is context free if and only if some pushdown automaton recognizes it

- As usual with “if and only if” theorems, there are two directions to prove
  - If a language is context free, then some pushdown automaton recognizes it
  - If a pushdown automaton recognizes some language, then it is context free

Only If (CFG to PDA)

- Let $L = L(G)$ for some CFG $G = (V, \Sigma, P, S)$
- Idea: have PDA $A$ simulate leftmost derivations in $G$, where a left-sentential form (LSF) is represented by:
  1. The sequence of input symbols that $A$ has consumed from its input, followed by...
  2. As stack, top left-most

Example: If $(q, abcd, S) \not\xrightarrow{*} (q, cd, ABC)$, then the LSF represented is $abABC$

Moves of A

- Place $\$ and the start variable on the stack
- Repeat:
  - If a terminal $a$ is on top of the stack, then if the string is in the language there will be an $a$ waiting on the input. $A$ consumes $a$ from the input and pops it from the stack
    - The LSF represented doesn’t change!
  - If a variable $B$ is on top of the stack, then PDA $A$ has a choice of replacing $B$ on the stack by the body of any production with head $B$
    - Non-deterministic!
  - If $\$ is on top of the stack, enter the accept state and accept if all input has been read

Notation

$(r, xyz) \in \delta(q, a, s)$

- When
  - $q$ is the current state
  - $a$ is the next input symbol
  - $s$ is on the top of the stack
- Do the following:
  - Read $a$
  - Pop $s$
  - Push $xyz$
Defining the PDA

- Define PDA $A$ as follows:
  - $Q = \{q_{\text{start}}, q_{\text{loop}}, q_{\text{accept}}\} \cup E$
  - $E$ is the set of states needed to implement the notation
  - $q_{\text{loop}}$ is the start state
  - $\Sigma$ contains the terminal symbols of the grammar
  - $\Gamma$ contains all terminal and non-terminal symbols from the grammar
  - $F = \{q_{\text{accept}}\}$

- $d$ is defined as follows:
  - For each production $X \rightarrow a$ in the grammar, create a move $d(q_{\text{loop}}, e, X) = (q_{\text{loop}}, a)$
  - For each terminal symbol $a$ in the grammar, create a move $d(q_{\text{loop}}, a, a) = (q_{\text{loop}}, e)$
  - To handle $\$$ on the top of the stack, create a move $d(q_{\text{loop}}, e, \$$) = (q_{\text{accept}}, e)$

Example

$S \rightarrow a \mid aS \mid bSS \mid SSb \mid SbS$

PDA $A = (\{q_{\text{start}}, q_{\text{accept}}, q_{\text{loop}}, dirt\}, \{a, b\}, \{S, a, b\}, \delta, q_{\text{start}}, q_{\text{accept}}, 5)$

$\delta$ is defined as

$\delta(q_{\text{start}}, e, \$) = (q_{\text{loop}}, S)$

$\delta(q_{\text{loop}}, e, S) = \{(q_{\text{loop}}, a), (q_{\text{loop}}, aS), (q_{\text{loop}}, bSS), (q_{\text{loop}}, SSb)\}$

$\delta(q_{\text{loop}}, a, a) = (q_{\text{loop}}, d)$

$\delta(q_{\text{loop}}, b, b) = (q_{\text{loop}}, d)$

$\delta(q_{\text{loop}}, e, \$) = (q_{\text{accept}}, d)$

Processing of baa

<table>
<thead>
<tr>
<th>state</th>
<th>input</th>
<th>stack</th>
<th>move</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_{\text{start}}$</td>
<td>baa</td>
<td>-</td>
<td>$d(q_{\text{loop}}, e, baa) = (q_{\text{loop}}, S)$</td>
</tr>
<tr>
<td>$q_{\text{loop}}$</td>
<td>baa</td>
<td>$SS$</td>
<td>$d(q_{\text{loop}}, e, S) = (q_{\text{loop}}, bSS)$</td>
</tr>
<tr>
<td>$q_{\text{loop}}$</td>
<td>aa</td>
<td>$SSS$</td>
<td>$d(q_{\text{loop}}, e, a) = (q_{\text{loop}}, d)$</td>
</tr>
<tr>
<td>$q_{\text{loop}}$</td>
<td>aa</td>
<td>$aS$</td>
<td>$d(q_{\text{loop}}, e, a) = (q_{\text{loop}}, d)$</td>
</tr>
<tr>
<td>$q_{\text{loop}}$</td>
<td>a</td>
<td>$S$</td>
<td>$d(q_{\text{loop}}, e, a) = (q_{\text{loop}}, d)$</td>
</tr>
<tr>
<td>$q_{\text{loop}}$</td>
<td>e</td>
<td>-</td>
<td>$d(q_{\text{loop}}, e, $) = (q_{\text{accept}}, d)$</td>
</tr>
<tr>
<td>$q_{\text{accept}}$</td>
<td>-</td>
<td>-</td>
<td>- accept -</td>
</tr>
</tbody>
</table>

Input: $b\ a\ a$

Try it

$R \rightarrow XRX \mid S$

$S \rightarrow aTa \mid bTa$

$T \rightarrow XTX \mid X \mid \epsilon$

$X \rightarrow a \mid b$
The PDA

\[ N = \{(q_{start}, q_{accept}, q_{loop}), \{a, b\}, \{a, b, R, S, T, X\}, \delta, q_{start}, q_{accept}, R\} \]

\[ P = \delta(q_{start}, a, \varepsilon) = \{q_{loop}, R\} \]
\[ \delta(q_{loop}, a, \varepsilon, R) = \{q_{loop}, XRX, (q_{loop}, S)\} \]
\[ \delta(q_{loop}, a, T) = \{(q_{loop}, XTX, (q_{loop}, X), (q_{loop}, \varepsilon)\} \]
\[ \delta(q_{loop}, a, X) = \{(q_{loop}, a), (q_{loop}, b)\} \]
\[ \delta(q_{loop}, a, a) = (q_{loop}, \varepsilon) \]
\[ \delta(q_{loop}, b, b) = (q_{loop}, \varepsilon) \]
\[ \delta(q_{loop}, a, \varepsilon) = (q_{accept}, \varepsilon) \]

Using the PDA

Using the PDA

Using the PDA
Converting from PDA to CFG

- A PDA *consumes* a character
- A CFG *generates* a character
- We want to relate these two
- What happens when a PDA consumes a character?
  - It may change state
  - It may change the stack

PDA to CFG

- Assume $L = N(P)$, where $P = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{accept}})$,
  - $q_{\text{accept}}$ is empty (accept by empty stack)
  - Start variable is $A_{q_{\text{start}}}q_{\text{accept}}$
- Key idea: units of PDA action have the net effect of popping one symbol from the stack, consuming some input, and making a state change
- The CFG variable $A_{q,p}$ generates exactly those strings $w$ such that $P$ can read $w$ from the input, pop one symbol from the stack, and go from state $q$ to state $p$
  - More precisely, $(q, w, Z) \not\rightarrow (p, \varepsilon, \varepsilon)$
  - As a consequence of above, $(q, wx, Z\alpha) \not\rightarrow (p, x, \alpha)$ for any $x$ and $\alpha$

Converting from PDA to CFG

- Suppose $X$ is on the stack and $a$ is read
- What can happen to $X$?
  - It can be popped
  - It may be replaced by one or more other stack symbols
  - And so on…
  - The stack grows and shrinks and grows and shrinks…
  - Eventually, as more input is consumed, $X$ must be popped (or we’ll never reach an empty stack)
  - And the state may change many times
  - We must track all of this!

*It’s a Zen thing*

$A_{q,p}$ is at once a variable involving states and symbols of $P$, and yet to the CFG we construct, it is a single, indivisible object

(OK, I know that’s not a Zen thing, but you get the point)