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 \( (q, abcd, S) \vdash^*(q, cd, ABC) \), 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. PDA \( A \) 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 \( A \) 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**

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

- \(q\) 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 \(xyz\)

**Example**

\(S \rightarrow a \mid aS \mid bSS \mid SSb \mid SbS\)

PDA \(A = \langle\{q_{\text{start}}, q_{\text{accept}}, q_{\text{loop}}\}, \{a, b\}, \{S, a, b\}, \delta, q_{\text{start}}, q_{\text{accept}}, S\rangle\)

\(\delta\) is defined as

\[
\begin{align*}
\delta(q_{\text{start}}, \varepsilon, \varepsilon) &= (q_{\text{loop}}, S) \\
\delta(q_{\text{loop}}, \varepsilon, S) &= \{(q_{\text{loop}}, a), (q_{\text{loop}}, aS), (q_{\text{loop}}, bSS), (q_{\text{loop}}, SSb), (q_{\text{loop}}, SbS)\} \\
\delta(q_{\text{loop}}, a, a) &= (q_{\text{loop}}, \varepsilon) \\
\delta(q_{\text{loop}}, b, b) &= (q_{\text{loop}}, \varepsilon) \\
\delta(q_{\text{loop}}, \varepsilon, \$) &= (q_{\text{accept}}, \varepsilon)
\end{align*}
\]

**Defining the PDA**

Define PDA \(A\) as follows:

- \(Q = \{q_{\text{start}}, q_{\text{loop}}, q_{\text{accept}}\} \cup \mathcal{E}\)
- \(E\) is the set of states needed to implement the notation
- \(q_{\text{start}}\) is the start state
- \(\Sigma\) contains the terminal symbols of the grammar
- \(\Gamma\) contains all terminal and non-terminal symbols from the grammar
- \(F = q_{\text{accept}}\)

\(\delta\) is defined as follows:

- For each production \(X \rightarrow \alpha\) in the grammar, create a move 
  \(\delta(q_{\text{loop}}, \varepsilon, X) = (q_{\text{loop}}, \alpha)\)
- For each terminal symbol \(a\) in the grammar, create a move
  \(\delta(q_{\text{loop}}, a, a) = (q_{\text{loop}}, \varepsilon)\)
- To handle \(\$\) on the top of the stack, create a move
  \(\delta(q_{\text{loop}}, \varepsilon, \$) = (q_{\text{accept}}, \varepsilon)\)

**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>(\varepsilon)</td>
<td>(\delta(q_{\text{loop}}, \varepsilon, \varepsilon) = (q_{\text{loop}}, S$))</td>
</tr>
<tr>
<td>(q_{\text{loop}})</td>
<td>(baa)</td>
<td>(S$)</td>
<td>(\delta(q_{\text{loop}}, \varepsilon, S) = (q_{\text{loop}}, bSS))</td>
</tr>
<tr>
<td>(q_{\text{loop}})</td>
<td>(baa)</td>
<td>(bSS$)</td>
<td>(\delta(q_{\text{loop}}, b, b) = (q_{\text{loop}}, \varepsilon))</td>
</tr>
<tr>
<td>(q_{\text{loop}})</td>
<td>(aa)</td>
<td>(SS$)</td>
<td>(\delta(q_{\text{loop}}, \varepsilon, S) = (q_{\text{loop}}, a))</td>
</tr>
<tr>
<td>(q_{\text{loop}})</td>
<td>(aa)</td>
<td>(aSS$)</td>
<td>(\delta(q_{\text{loop}}, a, a) = (q_{\text{loop}}, \varepsilon))</td>
</tr>
<tr>
<td>(q_{\text{loop}})</td>
<td>(a)</td>
<td>(a$)</td>
<td>(\delta(q_{\text{loop}}, \varepsilon, a) = (q_{\text{loop}}, \varepsilon))</td>
</tr>
<tr>
<td>(q_{\text{loop}})</td>
<td>(\varepsilon)</td>
<td>(\varepsilon$)</td>
<td>(\delta(q_{\text{loop}}, \varepsilon, \varepsilon) = (q_{\text{accept}}, \varepsilon))</td>
</tr>
<tr>
<td>(q_{\text{accept}})</td>
<td></td>
<td></td>
<td>accept</td>
</tr>
</tbody>
</table>

Input: \(b \rightarrow a \rightarrow a\)
Using the PDA

(q, aabbba, R) ∈ (q, aabbba, XRX)
(q, aabbba, aRX) ⊢ (q, aabbba, aRX)
(q, aabbba, RX) ⊢ (q, aabbba, RX)
(q, aabbba, SX) ⊢ (q, aabbba, SX)
(q, aabbba, aTbX) ⊢ (q, aabbba, aTbX)
(q, bba, TbX) ⊢ (q, bba, TbX)
(q, bba, XbX) ⊢ (q, bba, XbX)
(q, bba, bbX) ⊢ (q, bba, bbX)
(q, ba, bX) ⊢ (q, ba, bX)
(q, a, X) ⊢ (q, a, X)
(q, a, a) ⊢ (q, a, a)
(q, ε, ε) ⊢ (q, ε, ε)

δ(q, ε, R) = (q, XRX)
δ(q, ε, X) = (q, a)
δ(q, a, a) = (q, ε)
δ(q, ε, R) = (q, S)
δ(q, ε, S) = (q, aTb)
δ(q, a, a) = (q, ε)
δ(q, ε, X) = (q, b)
δ(q, b, b) = (q, ε)
δ(q, ε, X) = (q, a)
δ(q, ε, a) = (q, ε)

The PDA

\begin{align*}
N &= \{q_{\text{start}}, q_{\text{accept}}, q_{\text{loop}}, \{a, b\}, \{a, b, R, S, T, X\}, \delta, q_{\text{start}}, q_{\text{accept}}\} \\
P &= \delta(q_{\text{start}}, \epsilon, \epsilon) = \{(q_{\text{loop}}, R\$)\} \\
\delta(q_{\text{loop}, \epsilon, R}) &= \{(q_{\text{loop}}, XRX), (q_{\text{loop}}, S)\} \\
\delta(q_{\text{loop}, \epsilon, S}) &= \{(q_{\text{loop}}, aTb), (q_{\text{loop}, \epsilon}, bTb)\} \\
\delta(q_{\text{loop}, \epsilon, T}) &= \{(q_{\text{loop}}, XTX), (q_{\text{loop}, \epsilon}, X), (q_{\text{loop}, \epsilon}, \epsilon)\} \\
\delta(q_{\text{loop}, \epsilon, X}) &= \{(q_{\text{loop}, \epsilon}, a), (q_{\text{loop}, \epsilon}, b)\} \\
\delta(q_{\text{loop}, a, a}) &= (q_{\text{loop}, \epsilon}) \\
\delta(q_{\text{loop}, b, b}) &= (q_{\text{loop}, \epsilon}) \\
\delta(q_{\text{loop}, \epsilon, \$}) &= (q_{\text{accept, \epsilon}}) \\
\end{align*}
Using the PDA

**Matched**

(q, aabba, R) \(\vdash\) (q, aabba, XRX)
(q, aabba, XRX) \(\vdash\) (q, aabba, aRX)
(q, aabba, aRX) \(\vdash\) (q, aabba, RX)
a (q, abba, RX) \(\vdash\) (q, abba, SX)
a (q, abba, SX) \(\vdash\) (q, abba, aTbX)
a (q, abba, aTbX) \(\vdash\) (q, abba, TbX)
ba (q, bba, TbX) \(\vdash\) (q, bba, XbX)
ba (q, bba, XbX) \(\vdash\) (q, bba, bbX)
ba (q, bba, bbX) \(\vdash\) (q, ba, bX)
bba (q, bba, bX) \(\vdash\) (q, a, X)
bba (q, a, X) \(\vdash\) (q, a, a)
bba (q, a, a) \(\vdash\) (q, a, ε)
bba (q, ε, ε)

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

Converting from PDA to CFG

**continued**

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!

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, \((q, w, Z) \vdash^* (p, \epsilon, \epsilon)\)

As a consequence of above, \((q, wx, Z\alpha) \vdash^* (p, x, \alpha)\) for any \(x\) and \(\alpha\)
$A_{q,p}$ is at once a variable involving states and symbols of $P$, and yet to the CFG we construct, it is a single, indivisible object.