Push Down Automata

Push Down Automata (PDAs) are $\Lambda$-NFAs with stack memory.

Transitions are labeled by an input symbol together with a pair of the form $X/\alpha$.

The transition is possible only if the top of the stack contains the symbol $X$.

After the transition, the stack is changed by replacing the top symbol $X$ with the string of symbols $\alpha$. (Pop $X$, then push symbols of $\alpha$.)
Example

PDAs can accept languages that are not regular. The following one accepts:

$L = \{0^i1^j \mid 0 \leq i \leq j\}$

![Diagram of a PDA accepting $L$]
Definition

A PDA is a 7-tuple $P= (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ where $Q$, $\Sigma$, $q_0$, $F$ are as in NFAs, and

- $\Gamma$ is the *stack alphabet*.

- $Z_0 \in \Gamma$ is the *start symbol*; it is assumed that initially the stack contains only the symbol $Z_0$.

- $\delta : Q \times (\Sigma \cup \{\Lambda\}) \times \Gamma \longrightarrow P(Q \times \Gamma^*)$ is the *transition function*: given a state, an input symbol (or $\Lambda$), and a stack symbol, it gives us a finite number of pairs $(q, \alpha)$, where $q$ is the next state and $\alpha$ is the string of stack symbols that will replace $X$ on top of the stack.
In our example, the transition from $s$ to $s$ labeled $(0, Z_0/XZ_0)$ corresponds to the fact $(s, XZ_0) \in \delta(s, 0, Z_0)$. A complete description of the transition function in this example is given by

\[
\begin{align*}
\delta(s,0,Z_0) &= \{(s,XZ_0)\} \\
\delta(s,0,X) &= \{(s,XX)\} \\
\delta(s,\Lambda,Z_0) &= \{(q,Z_0)\} \\
\delta(s,1,X) &= \{(p,\varepsilon)\} \\
\delta(p,1,X) &= \{(p,\varepsilon)\} \\
\delta(p,\Lambda,Z_0) &= \{(q,Z_0)\} \\
\delta(q,1,Z_0) &= \{(q,Z_0)\} \\
\text{and} \\
\delta(q,a,Y) &= \emptyset \\
\text{for all other possibilities.}
\end{align*}
\]
Instantaneous Descriptions and Moves of PDAs

IDs (also called *configurations*) describe the execution of a PDA at each instant. An ID is a triple \((q, w, \alpha)\), with this intended meaning:

- \(q\) is the current state
- \(w\) is the remaining part of the input
- \(\alpha\) is the current content of the stack, with top of the stack on the left.
The relation $\rightarrow$ describes possible moves from one ID to another during execution of a PDA. If $\delta(q, a, X)$ contains $(p, \alpha)$, then

$$(q, aw, X\beta) \rightarrow (p, w, \alpha\beta)$$

is true for every $w$ and $\beta$.

The relation $\rightarrow^*$ is the reflexive-transitive closure of $\rightarrow$.

We have $(q, w, a) \rightarrow^*(q', w', a')$ when $(q, w, a)$ leads through a sequence (possibly empty) of moves to $(q', w', a')$. 
\[(s,011,z) \rightarrow (s,11,xz) \rightarrow (P,1,z) \rightarrow (q,1,z) \rightarrow (q,,,Z)\]

\[(s,011,z) \rightarrow (q,011,Z)\]
Properties of |- *

**Property 1.**
If \( (q, x, \alpha) \ |-* (p, y, \beta) \)  
Then \( (q, xw, \alpha\gamma) \ |-* (p, yw, \beta\gamma) \)

If you only need some prefix of the input \( x \) and stack \( \alpha \) to make a series of transitions, you can make the same transitions for any longer input and stack.

**Property 2.**
If \( (q, xw, \alpha) \ |-* (p, yw, \beta) \)  
Then \( (q, x, \alpha) \ |-* (p, y, \beta) \)

It is ok to remove unused input, since a PDA cannot add input back on once consumed.
A PDA as above *accepts* the string $w$ iff
$(q_0, w, Z_0) \vdash^* (p, \Lambda, \alpha)$ is true for some final state $p$ and some $\alpha$. (We don't care what's on the stack at the end of input.)

The *language* $L(P)$ of the PDA $P$ is the set of all strings accepted by $P$. 
Here is the chain of IDs showing that the string 001111 is accepted by our example PDA:

\[(s,001111,Z_0)\]
\[- (s,01111,XZ_0)\]
\[- (s,1111,XXZ_0)\]
\[- (p,111,XZ_0)\]
\[- (p,11,Z_0)\]
\[- (q,11,Z_0)\]
\[- (q,\epsilon,Z_0)\]
\[- (q,\epsilon,Z_0)\]
The language of the following PDA is \( \{0^i1^j \mid 0 < i \leq j\}^* \).
How can we prove this?
Example

A PDA for the language of balanced parentheses:

\[ (, z_0/xz_0 \]
\[ (, x/xx \]
\[ (, x/\varepsilon \]

\[ \Lambda, z_0/z_0 \]

\[ p \rightarrow q \]
Acceptance by Empty Stack

Define \( N(P) \) to be the set of all strings \( w \) such that
\[
(q_0, w, Z_0) \rightarrow^* (q, \Lambda, \epsilon)
\]
for some state \( q \). These are the strings \( P \) accepts by empty stack. Note that the set of final states plays no role in this definition.

**Theorem.** A language is \( L(P_1) \) for some PDA \( P_1 \) if and only if it is \( N(P_2) \) for some PDA \( P_2 \).
Proof 1

1. From empty stack to final state.

Given $P_2$ that accepts by empty stack, get $P_1$ by adding a new start state and a new final state as in the picture below. We also add a new stack symbol $X_0$ and make it the start symbol for $P_1$'s stack.

(add this transition from all states of $P_2$ to new state $P_f$)
2. From final state to empty stack.

Given $P_1$, we get $P_2$ again by adding a new start state, final state and start stack symbol. New transitions are seen in the picture.
Equivalence of CFGs and PDAs

The equivalence is expressed by two theorems.

**Theorem 1.** Every context-free language is accepted by some PDA.

**Theorem 2.** For every PDA $M$, the language $L(M)$ is context-free.

We will describe the constructions, see some examples and proof ideas.
Given a CFG $G=(V,T,P,S)$, we define a PDA $M=\{q\}, T, T \cup V, \delta, q, S$, with $\delta$ given by:

- If $A \in V$, then $\delta(q, \varepsilon, A) = \{ (q, \alpha) \mid A \rightarrow \alpha \text{ is in } P \}$
- If $a \in T$, then $\delta(q, a, a) = \{ (q, \varepsilon) \}$

1. Note that the stack symbols of the new PDA contain all the terminal and non-terminals of the CFG
2. There is only 1 state in the new PDA, all the rest of the info is encoded in the stack.
3. Most transitions are on $\Lambda$, one for each production
4. The other transitions come one for each terminal.

The automaton simulates leftmost derivations of $G$, accepting by empty stack. For every intermediate sentential form $uA_\alpha$ in the leftmost derivation of $w$ (note first that $w = uv$ for some $v$), $M$ will have $A_\alpha$ on its stack after reading $u$. At the end (case $u = w$) the stack will be empty.
Example

For our old grammar: \( S \rightarrow SS | (S) | \Lambda \)

the automaton \( M \) will have five transitions, all from \( q \) to \( q \):

1. \( \delta(q, \Lambda, S) = (q, SS) \)  \( S \rightarrow SS \)
2. \( \delta(q, \Lambda, S) = (q, (S)) \)  \( S \rightarrow (S) \)
3. \( \delta(q, \Lambda, S) = (q, \Lambda) \)  \( S \rightarrow \Lambda \)
4. \( \delta(q, (, ( ) = (q, \Lambda) \)
5. \( \delta(q, ), ) = (q, \Lambda) \)

1. Most transitions are on \( \Lambda \), one for each production
2. The other transitions come one for each terminal.
Now compare the leftmost derivation
\[ S \Rightarrow SS \Rightarrow (S)S \Rightarrow ((S))S \Rightarrow ((()))S \Rightarrow ((()))(S) \Rightarrow ((()))() \]
with the M's execution on the same string given as input:

1. \( \delta(q, \Lambda, S) = (q, SS) \) \( S \rightarrow SS \)
2. \( \delta(q, \Lambda, S) = (q, (S)) \) \( S \rightarrow (S) \)
3. \( \delta(q, \Lambda, S) = (q, \varepsilon) \) \( S \rightarrow \Lambda \)
4. \( \delta(q, (, ( )) = (q, \Lambda) \)
5. \( \delta(q, ), ) ) = (q, \Lambda) \)
Next time

We’ll prove the construction correct,

Look at the inverse construction. PDA→CFL