1 Some Properties of Pushdown Automata

- Pushdown Automata (PDA) recognize context free languages.
- These automata are like nondeterministic finite state automata but have an extra component called a stack.
  It is this extra component that provides the automaton with memory (in principle, an infinite amount of memory), and allows it to recognize some nonregular languages.
- A PDA can write (push) a symbol on the top of the stack or remove (pop) a symbol from the top of the stack.
  The stack is, in principle, unlimited, and it works as a last in, first out storage device.
- Recall that \( \{0^n1^n \mid n \geq 0\} \) cannot be recognized by a finite state automaton (FSA). But a PDA can recognize this language with the help of the stack.

  1. As the machine reads a 0 from the input string, push it on top of the stack.
  2. As soon as 1s are encountered from the input string, pop a 0 off the stack for each 1 read.
  3. If reading the input string is finished exactly when the stack becomes empty of 0s, accept the input.
     Reject otherwise.

- Conditions under which the input string is accepted or rejected by a PDA:

  1. ACCEPT the input string if the stack is empty when the last symbol is read.
  2. REJECT otherwise.

2 Formal Definition of Pushdown Automata

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. \( \delta : Q \times \Sigma_e \times \Gamma_e \rightarrow \varphi(Q \times \Gamma_e) \) is the transition function,

5. \( q_0 \in Q \) is the start state, and

6. \( F \subseteq Q \) is the set of accept states.

A PDA \( M = \langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle \) computes a string \( w \) as follows. It accepts an input string \( w \):

- if \( w \) can be written as \( w = w_1 w_2 ... w_m \), where each \( w_i \in \Sigma_e \), and

- there is a sequence of states \( r_0, r_1, ..., r_m \in Q \) and strings \( s_0 s_1, ..., s_m \in \Gamma^* \) that satisfy the following conditions:

  1. \( r_0 = q_0 \) and \( s_0 = \epsilon \). This condition requires that \( M \) start out in its start state and with an empty stack.

  2. For \( i = 0, ... m - 1 \), we have \( (r_{i+1}, b) \in \delta(r_i, w_{i+1}, a) \), where:

      - \( s_i = at \)
      - \( s_{i+1} = bt \)

    for some \( a, b \in \Gamma_e \) and \( t \in \Gamma^* \).

    That is, each move of \( M \) is performed according to the state, the contents of the stack and the next input symbol.

  3. \( r_m \in F \).

This tells us that the last state in the computation must be an accept state.

3 Examples of Pushdown Automata

- PDA that recognizes the language \( \{0^n1^n \mid n \geq 0\} \).

Let \( M_1 = \langle Q, \Sigma, \Gamma, \delta, q_1, F \rangle \), where

\( Q = \{q_1, q_2, q_3, q_4\} \),

\( \Sigma = \{0, 1\} \),

\( \Gamma = \{0, \$\} \),

\( F = \{q_1, q_4\} \), and

\( \delta \) is given by the following table:

<table>
<thead>
<tr>
<th>Input</th>
<th>Stack</th>
<th>0</th>
<th>$</th>
<th>\epsilon</th>
<th>1</th>
<th>0</th>
<th>$</th>
<th>\epsilon</th>
<th>\epsilon</th>
</tr>
</thead>
<tbody>
<tr>
<td>q1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>{(q2,$)}</td>
</tr>
<tr>
<td>q2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>{(q2,0)}</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>q3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>{(q3,\epsilon)}</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>{(q4,\epsilon)}</td>
<td>-</td>
</tr>
<tr>
<td>q4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\( \delta \) is given by the following table:

- **Input:** 0, \( e \rightarrow 0 \)
- **Input:** 1, \( e \rightarrow \epsilon \)
- **Input:** \( e, \epsilon \rightarrow \$ \)
- **Input:** \( e, \epsilon \rightarrow e \)
- **Input:** 1, \( 0 \rightarrow e \)
- **Input:** \( e, \$ \rightarrow e \)
Notations:

- \( a, b \rightarrow c \): when the machine is reading an \( a \) from the input, it may replace the symbol \( b \) on the top of the stack with a \( c \) (pop \( b \) and push \( c \)).
  Any of \( a, b \) and \( c \) may be \( \epsilon \):
  - If \( a \) is \( \epsilon \), the machine may make this transition without reading any symbol from the input.
  - If \( b \) is \( \epsilon \), the machine may make this transition without popping a symbol from the stack (pop nothing and push \( c \)).
  - If \( c \) is \( \epsilon \), the machine does not write any symbol on the stack in this transition (pop \( b \) and push nothing).

- \( \epsilon \rightarrow \$ \) places a special symbol \( \$ \) on the stack. This mechanism allows the PDA to test for an empty stack. By initially placing \$ on the stack, the machine knows that the stack is effectively empty when it sees the \$ again.

- **Example 1:** A PDA that recognizes the language \( \{a^ib^jc^k| i, j, k \geq 0 \text{ and } i = j \text{ or } i = k \} \).

The formal description of this machine is given as follows:

Let \( M = \langle Q, \Sigma, \Gamma, \delta, q_1, F \rangle \), where

- \( Q = \{q_1, q_2, q_3, q_4, q_5, q_6, q_7\} \),
- \( \Sigma = \{a, b, c\} \),
- \( \Gamma = \{a, \$\} \),
- \( F = \{q_4, q_7\} \), and
- \( \delta \) is given by the following table:

<table>
<thead>
<tr>
<th>State</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_1 )</td>
<td>0</td>
<td>0</td>
<td>$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>0</td>
<td>0</td>
<td>{q_2,a}</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>0</td>
<td>0</td>
<td>{q_3,c}</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>{q_4,c}</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( q_5 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>{q_5,c}</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( q_6 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>{q_6,a}</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( q_7 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>{q_7,a}</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Example 2: A PDA that recognizes the language \( \{ w w^R | w \in \{ 0, 1 \}^* \} \).

![Diagram of a PDA]

Question: Give a formal description of this machine.

4 Equivalence with Context Free Grammars

Context free grammars (CFGs) and pushdown automata (PDA) are equivalent in power. That is, any CFG can be converted into a PDA that recognizes the same language and vice versa.

**Theorem (Sipser’s Theorem 2.12)**

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

5 Closure Properties of Context Free Languages


5.1 Union

Given two CFG’s \( G_1 = \langle V_1, \Sigma, R_1, S_1 \rangle \), and \( G_2 = \langle V_2, \Sigma, R_2, S_2 \rangle \), we form \( G \) (where \( G \) accepts \( L(G_1) \cup L(G_2) \)) in the following way.

- If the variables of \( G_1 \) and \( G_2 \) are not disjoint sets, we make them so (by appending primes to every variable of \( G_2 \)).
• The start symbol of \( G \) we take to be \( S \), and \( G \) contains, in addition to \( R1 \) and \( R2 \), the rules \( S \to S1 \) and \( S \to S2 \).

• \( G \) is context free, and it generates \( L(G1) \cup L(G2) \) since the start symbol may be either rewritten as \( S1 \), whereupon \( G \) behaves like \( G1 \), or as \( S2 \), whereupon \( G \) behaves like \( G2 \).

• Question: Construct \( G \) that generates \( L(G1) \cup L(G2) \), where \( G1 \) and \( G2 \) are as follows:

\[
\begin{align*}
G1 & \\
S & \to BC \\
B & \to aBb \\
B & \to \epsilon \\
C & \to cC \\
C & \to \epsilon \\
\end{align*}
\[
\begin{align*}
G2 & \\
S & \to BC \\
B & \to aB \\
B & \to \epsilon \\
C & \to bCc \\
C & \to \epsilon \\
\end{align*}
\]

5.2 Concatenation

Given two CFG’s \( G1 = \langle V1, \Sigma, R1, S1 \rangle \), and \( G2 = \langle V2, \Sigma, R2, S2 \rangle \), we form \( G \) in the following way.

• If the variables of \( G1 \) and \( G2 \) are not disjoint sets, we make them so (by appending primes to every variable of \( G2 \)).

• The start symbol of \( G \) we take to be \( S \), and \( G \) contains, in addition to \( R1 \) and \( R2 \), the rule \( S \to S1S2 \).

• \( G \) will generate all and only strings of the from \( xy \) such that \( x \in L(G1) \) and \( y \in L(G2) \).

• Question: Construct \( G \) that generates \( L(G1) \circ L(G2) \), where \( G1 \) and \( G2 \) are defined as above.
5.3 Star Operation  

Given $G = \langle V, \Sigma, R, S \rangle$, we construct $G^*$ in the following way.

- The start symbol of $G^*$ is $S'$, and $G^*$ contains, in addition to all the rules in $R$, the rules $S' \rightarrow \epsilon$ and $S' \rightarrow S'S$.

- $G^*$ generates all strings in $(L(G))^*$ since by application of the rules rewriting $S'$, $G^*$ produces strings $S^n$ for all $n \geq 0$. Each such $S$ can be rewritten to produce a string in $L(G)$, and $\epsilon$ is produced by the rule $S' \rightarrow \epsilon$.

- Question: Construct $G1^*$ that generates $(L(G1))^*$, where $G1$ is defined as above.

5.4 Intersection  

CFLs are NOT closed under intersection.

- To see this, we note that the languages $L1 = \{a^i b^j c^j \mid i, j \geq 0 \}$ and $L2 = \{a^k b^l c^{k+l} \mid k, l \geq 0 \}$ are both context free (see Example 1 above).

$L1$ is generated by the following CFG:

\[
\begin{align*}
S & \rightarrow BC \\
B & \rightarrow aBb \\
B & \rightarrow \epsilon \\
C & \rightarrow cC \\
C & \rightarrow \epsilon 
\end{align*}
\]

$L2$ is generated by the following CFG:

\[
\begin{align*}
S & \rightarrow BC \\
B & \rightarrow aB \\
B & \rightarrow \epsilon \\
C & \rightarrow bCc \\
C & \rightarrow \epsilon 
\end{align*}
\]

$L1 \cap L2 = \{a^n b^n c^n \mid n \geq 0 \}$.

But this is not a context free language. (We will see why shortly.)
5.5 Complementation

CFLs are not closed under complementation.

- Given two CFLs $L_1$ and $L_2$ over some alphabet $\Sigma$, if their complements $L_1' = \Sigma^* - L_1$ and $L_2' = \Sigma^* - L_2$ were context free, then so would be the union of those complements, $L_1' \cup L_2'$.

- The complement of this, in turn, $(L_1' \cup L_2')'$ would also be context free. This is equal to $(L_1 \cap L_2)$, by DeMorgan’s Laws.

- But $(L_1 \cap L_2)$ is not necessarily context free (as we saw in the previous section).

6 The Pumping Lemma for Context Free Languages

6.1 Theorem (Sipser’s Theorem 2.10)

If $A$ is a context free language, then there is a number $p$ (pumping length) where, if $s$ is any string in $A$ of length at least $p$, then $s$ may be divided into five pieces $s = uvxyz$ satisfying the following conditions:

1. For each $i \geq 0$, $uv^i xy^i z \in A$,

2. $|vxy| > 0$, and

   Either $v$ or $y$ is not the empty string. Otherwise the theorem would be trivially true.

3. $|vxy| \leq p$.

   The pieces $v$, $x$ and $y$ together have length at most $p$.

6.2 An example

- Let’s apply the pumping lemma to the following language $A$.

  $A = \{0^n1^n \mid n \geq 0\}$.

  Let’s assume that the pumping length is 5. Let’s take some string longer than 5. How about 000111? We can then break this string down as follows:

  $$u = 0, v = 0, x = 01, y = 1, z = 1$$

  By pumping $v$ and $y$, we get:

  $uv^p xy^p z = 0-01-1$

  $uv^3 xy z = 0-0-01-1-1$

  $uv^2 xy^2 z = 0-00-01-11-1$

  $uv^3 xy^3 z = 0-000-01-111-1$

  etc.
All of these strings are in language $A$. So, the pumping lemma works for this language and this string.

**Important:** The pumping lemma says "if a language is context free, then there is a number $p$ such that ...". Let's represent this theorem as $c \rightarrow p$. We use the contrapositive to show that a language is not context free: $\neg p \rightarrow \neg c$. If, on the other hand, we simply show that the pumping lemma works for a language, we cannot conclude on this basis that the language is context free. That is, given $c \rightarrow p$, if we show that $p$ holds we have not thereby demonstrated the truth of $c$.

### 6.3 Proving that a language is not context free

See also Partee pages 494-5.

#### 6.3.1 Sipser’s Example 2.20

Use the pumping lemma to show that the language $B = \{a^n b^n c^n | n \geq 0\}$ is not context free.

We will do this by assuming that $B$ is context free, and showing that a contradiction follows from this assumption. Therefore, the assumption we started out with must be wrong, and thus $B$ is not context free.

- Let $p$ be the pumping length given by the pumping lemma. Choose $s$ to be the string $a^p b^p c^p$.

- Because $s \in B$ and $s$ has length greater than $p$, the pumping lemma guarantees that we can split $s$ into five pieces, $s = uvxyz$ in such a way that for any $i \geq 0$, the string $uv^i xy^i z$ is in $B$. But no matter how we divide $s$ into $uvxyz$, one of the three conditions of the lemma is violated. We consider two cases to show this:

  1. When both $v$ and $y$ consist of one or more instances of a single symbol of the alphabet, $v$ does not contain both $a$’s and $b$’s or both $b$’s and $c$’s, and the same holds for $y$. In this case, the string $uv^2 xy^2 z$ cannot contain equal numbers of $a$’s, $b$’s, and $c$’s. Therefore, it cannot be a member of $B$. This violates condition 1 of the lemma and is thus a contradiction.

  2. When either $v$ or $y$ contain more than one type of symbol $uv^2 xy^2 z$ may contain equal number of the three alphabet symbols but won’t contain them in the correct order. Hence it cannot be a member of $B$ and a contradiction occurs.

  3. There is no other way to split up the string $s$, so a contradiction is unavoidable if we make the assumption that $B$ is context free. Thus, $B$ is not context free.

#### 6.3.2 Sipser’s Example 2.22

Use the pumping lemma to show that the language $C = \{ww \mid w \in \{0, 1\}^n\}$ is not context free.

- Assume to the contrary that $C$ is context free. Let $p$ be the pumping length given by the pumping lemma. Let $s$ be the string $0^p 1^p 0^p 1^p p$. 
The pumping lemma guarantees that we can pump $s$ by dividing it to $uvxyz$, where the substring $|vxy| \leq p$ according to condition 3 of the pumping lemma.

The substring $vxy$ must straddle the midpoint of $s$.

If the substring occurs only in the first half of $s$, pumping $s$ up to $uv^2xy^2z$ moves a 1 into the first position of the second half, and so it cannot be of the form $ww$.

If $vxy$ occurs in the second half of $s$, pumping $s$ up to $uv^2xy^2z$ moves a 0 into the last position of the first half, and so it cannot be of the form $ww$.

But if the substring $vxy$ straddles the midpoint of $s$, when we try to pump $s$ down to $uv^pxy^pz = uxz$, it has the form $0^i1^j0^p$, where $i$ and $j$ cannot both be $p$. This string is not of the form $ww$.

Thus $s$ cannot be pumped, and $C$ is not context free.

7 Are Natural Languages Context Free?

See Partee pages 501-3.

- Dutch and Swiss German have ‘crossing dependency’ structures.

$$x_1x_2x_3\ldots x_n\ldots y_1y_2y_3\ldots y_n$$

- Example: Swiss German

  (1)  a. Jan säit das mer em Hans es huus hälfe aastrische  
       John said that we$_i$ Hans$_j$ the house helped$_i$ paint$_j$
       ‘John said that we helped Hans paint the house.’
  
       b. Jan säit das mer d’chind em Hans es huus länd hälfe aastrische  
       John said that we$_i$ the-children$_j$ Hans$_k$ the house let$_i$ help$_j$ paint$_k$
       ‘John said that we let the children help Hans paint the house.’

- Crossing dependency structures correspond to the language $\{ww \mid w \in \{0,1\}^*\}$.

  This language is not context free, as was proven using the pumping lemma (see Sipser’s Example 2.22 above).

  $\Rightarrow$ Natural languages are a bit beyond context free languages.
8 The Chomsky Hierarchy

<table>
<thead>
<tr>
<th>Regular Languages, Finite State Automata, Right Linear Grammars</th>
</tr>
</thead>
<tbody>
<tr>
<td>↓</td>
</tr>
<tr>
<td>Context Free Languages, Pushdown Automata, Context Free Grammars</td>
</tr>
<tr>
<td>↓</td>
</tr>
<tr>
<td>Context Sensitive Languages, Linear Bounded Automata, Context Sensitive Grammars</td>
</tr>
</tbody>
</table>