Cpts 317

Fri Feb 21

Topic: Pushdown Automata
Push-down Automata (PDA)

- PDA generalizes from deterministic finite automata (DFA) but have an extra component called a stack.

The stack provides additional memory beyond that available in NFAs.

The stack allows PDA to recognize some non-regular languages.
Regular Languages
- NFA

Context-free Languages
- PDA

- PDA can write symbols on the stack and read them back later.
- Writing a symbol “push, down” all the other symbols on the stack.
- Stack is a “Last in First Out” storage device.

States & transition functions
A stack is valuable because it can hold unlimited amount of information (enable handling unbounded state problem, as in recognizing the language \{0^n 1^n \mid n \geq 0\}).

Informal description of how PDA for the above language works:

Read symbols from the input. As each 0 is read, push on stack. As soon as 1s are seen, pop a 0 off the stack for each 1 read. If reading the input is finished exactly when the stack becomes empty of 0s, accept input.
If stack becomes empty write 1s remain or if the 1s are finished write the stack still contains 0s or if any 0s appear in the input following 1s, reject input.
**Formal Definition of PDA**

A PDA is a 6-tuple

\[(Q, \Sigma, \Gamma, S, q_0, F)\]

where \(Q, \Sigma, \Gamma, \text{ and } F\) 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. \(S: Q \times \Sigma \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma)\) is the transition function
5. \(q_0 \in Q\) is the start state
6. \(F \subseteq Q\) is the set of accept states
A pushdown automaton

\[ M = (Q, \Sigma, \Gamma, \delta, q_0, F) \]

Computes as follows.

It accepts input \( w \) if \( w \) can be written as \( w = w_1 w_2 \ldots w_m \), where each \( w_i \in \Sigma \) and sequence of states

\[ q_0, q_1, \ldots, q_m \in \mathcal{Q} \] and strings

\[ s_0, s_1, \ldots, s_m \in \Gamma^* \] exist that satisfy the following three conditions:

1. \( q_0 = q_0 \) and \( s_0 = \varepsilon \)

2. For \( i = 0, \ldots, m-1 \) we have

\[ (q_{i+1}, b) \in \delta(q_i, w_i, a) \] where

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

\( a, b \in \Gamma \) and \( t \in \Gamma^* \)

3. \( q_m \in F \).
Example 1

Design a PDA that recognizes the language \{0^n 1^n | n \geq 0 \}.

A couple of notations before presenting the PDA.

1) Testing for empty stack.
   * The formal definition of a PDA contains no explicit mechanism to allow the PDA to test for an empty stack.
   * The same effect can be attained by initially putting a special symbol (\$) on the stack.

2) Short-hand notation
   for (state - input - state) transitions
We write 

\[ a, b \rightarrow c \]

to signify that when the machine is reading an \( a \) from the input, it may replace the symbol \( b \) on the top of the stack with \( a \) \( \rightarrow c \).

* Any of \( a, b, \) and \( c \) may be \( E \).

- \( A = E \rightarrow \) machine may make this transition without reading any symbol from the input.
- \( b = E \rightarrow \) machine may make this transition without reading & popping any symbol from the input.
- \( C = E \rightarrow \) machine does not write any symbol on the stack when going along this transition.
Example 2

Design a PDA that recognizes the language
\[ \{ w w^R \mid w \in \{0,1\}^* \} \, . \]

Recall that \( w^R \) means \( w \) written backwards.

(This language is a palindrome).

How would you about designing the PDA?
Theorem

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