A PDA to test that the complement of a particular language is a CFL

Prakash Panangaden

12th November 2018

We want to describe *briefly and informally* a pushdown automaton that recognizes the context free language $\overline{L}$ where $L = \{ww | w \in \Sigma^*\}$.

If the length of a word is odd then it certainly is in $\overline{L}$; testing to see if the length is odd or even can be done by a DFA. Thus, the harder case is testing to see if a word w is in $\overline{L}$ when $|w|$ is even. In order to be in $\overline{L}$, there must be a pair of letters at positions i and j in w with $w_i \neq w_j$ and with $j = i + |w|/2$; *i.e.* i, j are at corresponding positions in the two halves of w. The idea is that the PDA must guess where i and j are and also where the mid-point is. Accordingly it stacks the input as it reads it. At some point it guesses that it is looking at position i and *it remembers in its finite-state memory* what the letter at this position is. At this point it keeps on reading input letters and popping the stack. At some point the stack becomes empty, at this point it will start stacking input letters again. Then it guesses where j is and it compares the letter at position j with the letter in its finite-state memory. Now if these letters are indeed mismatched it has to verify that the position is correct. In order to do this it reads the rest of the input and this time it pops the stack as it reads input letters. At the end of the input the stack musy be empty. Why does this work? Let the string be of the following form:

$$\underbrace{x_1 x_2 \ldots x_{i-1}}_{(i-1)\text{ letters}} x_i \overbrace{x_{i+1} \ldots x_n}^{(n-i)\text{ letters}} \underbrace{y_1 y_2 \ldots y_{i-1}}_{(i-1)\text{ letters}} y_i \overbrace{y_{i+1} \ldots y_n}^{(n-i)\text{ letters}} .$$

Now the machine has guessed that x_i and y_i are different letters and this is what it checks. However, crucially, these must be verified to be in *corresponding positions.* In this case, it means that we have to check that the length of the strings $x_1 \ldots x_{(i-1)} y_{(i+1)} \ldots y_n$ is the same length as the string $x_{(i+1)} \ldots x_n y_1 \ldots y_{(i-1)}$. This is exactly what the PDA does. It stacks $x_1 \ldots x_{(i-1)}$, checks it off against $x_{(i+1)} \ldots x_n y_1 \ldots y_{(i-1)}$. The latter string must be longer so the stack will empty out before the string $x_{(i+1)} \ldots x_n y_1 \ldots y_{(i-1)}$ is done. The PDA then switches to stacking the symbols from the rest of $x_{(i+1)} \ldots x_n y_1 \ldots y_{(i-1)}$ and finally checks off its stacked symbols against the last remaining piece of the input *i.e.* $y_{(i+1)} \ldots y_n$.