COMP330 Assignment 3

Ryan Sowa, ID: 260886668

1. (20 points) Prove that the intersection of a context-free language C and a *regular language* R over the same alphabet Σ is context-free.

   Proof. Since C is context free, there exists a PDA, P $= (Q_1, \Sigma, \Gamma, \delta_1, q_0, F_1)$ for C. Since R is regular, there exists a DFA, D $= (Q_2, \Sigma, \delta_2, x_0, F_2)$ for R.

   We can construct a new PDA which simulates the parallel execution of P and D, $P' = (Q_1 \times Q_2, \Sigma, \Gamma, \delta', (q_0, x_0), F_1 \times F_2)$, and which accepts $C \cap R$. To illustrate the transition states, we can analyse 2 options:

   Case 1: P has a transition $\delta_1(q_1, (a, b \to c)) = q_2$ and D has transition $\delta_2(p_1, a) = p_2$. Our transition state for P' would then be $\delta'((q_1, p_1), (a, b \to c)) = (q_2, p_2)$.

   Case 2: P has a transition $\delta_1(q_1, (\epsilon, b \to c)) = q_2$ and D has a single state, p_1. Our transition state for P' would then be $\delta'((q_1, p_1), (\epsilon, b \to c)) = (q_2, p_1)$.

   As shown above, accept states in P' would be the cross product of accept states in P and D. The start state would be a combination of the start states of P and D.

   Since we have shown that P' accepts $C \cap R$, the intersection of C and R must be context free.

   □

2. (20 points) Either prove that the following language is context-free, or prove that it is not context-free. Here $\Sigma = \{0, 1\}$.

   $$L = \{ww^R | w \in \{0,1\}^*\} \cap \{w |\ w \text{ has the same number of 0's and 1's}\}.$$

   Proof. Suppose by contradiction that L is context-free. This implies L must have a pumping length, P such that any string $S \in L$,where $|S| \geq P$, can be divided into 5 parts, $S = uvxyz$. Let $S = 1^P 0^P 0^P 1^P \in L$. We

need $S = uvxyz$ such that the following conditions hold: 1. $|vxy| \leq P$, 2. v or $y \neq \epsilon$, and 3. $uv^ixy^iz \in L, \forall i$.

We can divide the string into 5 parts as follows: 1. v contains only 0's and y contains only 1's, 2. v contains only 1's and y contains only 0's, 3. v and y contain only 0's, and 4. v and y contain only 1's. Without loss of generality, we will only consider cases 1 and 3, since case 2 is symmetrical to case 1, and case 4 is symmetrical to case 3.

Case 1: Let $v = 0^{\frac{P}{2}}$, $x = \epsilon$, and $y = 1^{\frac{P}{2}}$. Now consider $i = 2$. Our new string is $1^P0^P0^{\frac{P}{2}}0^{\frac{P}{2}}1^{\frac{P}{2}}1^{\frac{P}{2}}1^{\frac{P}{2}} = 1^P0^P0^{\frac{3P}{2}}1^{\frac{3P}{2}} = 1^P0^{\frac{5P}{2}}1^{\frac{3P}{2}} \notin L$.

Case 3: Let $x = 0$ and let vy be the first set of 0's (without loss of generality). Consider $i = 2$. Our new string is $1^P0^{2P}0^P1^P = 1^P0^{3P}1^P \notin L$.

Since we have analyzed all relevant cases (which produced strings *not* in L), we can conclude by the Pumping Lemma that L is NOT context-free.

□

3. (20 points) Either prove that the following language is context-free, or prove that it is not context-free. Here $\Sigma = \{a, b, c, d\}$.

   $L = \{w |\ w$ has the same number of a's and b's, and additionally the same number of c's and d's$\}$.

   For example, $accddb \in L$.

   Proof. Suppose by contradiction that L is context-free. This implies L must have a pumping length, P such that any string $S \in L$,where $|S| \geq P$, can be divided into 5 parts, $S = uvxyz$. Let $S = a^Pb^Pc^Pd^P \in L$. We need $S = uvxyz$ such that the following conditions hold: 1. $|vxy| \leq P$, 2. v or $y \neq \epsilon$, and 3. $uv^ixy^iz \in L, \forall i$.

   Case 1: v and y each only contain one type of symbol. Without loss of generality, let that symbol be a. Let $x = \epsilon$ and consider $i = 2$. Then, our new string is $a^Pa^Pb^Pc^Pd^P = a^{2P}b^Pc^Pd^P \notin L$.

   Case 2: v and y contain different symbols. Without loss of generality, let those two symbols be a and b, respectively. Let $x = \epsilon$ and let v be the second half of the a's and y be the first half of the b's. Now consider $i = 2$. Then, our new string is $a^{\frac{P}{2}}a^{\frac{P}{2}}a^{\frac{P}{2}}b^{\frac{P}{2}}b^{\frac{P}{2}}b^{\frac{P}{2}}c^Pd^P = a^{\frac{3P}{2}}b^{\frac{3P}{2}}c^Pd^P \notin L$.

   Case 3: Either v or y contains more than one type of symbol. Without loss of generality, let v contain a's and b's while y contains only b's. Let $x = \epsilon$ and let $v = a^{\frac{P}{2}}b^{\frac{P}{4}}$ and $y = b^{\frac{P}{4}}$. Now consider $i = 2$. Then, our new string is $a^{\frac{P}{2}}a^Pb^{\frac{P}{2}}b^{\frac{P}{2}}b^{\frac{P}{2}}c^Pd^P = a^{\frac{3P}{2}}b^{\frac{3P}{2}}c^Pd^P \notin L$.

   Note: The above cases apply for any two adjacent symbols (not just a and b), since they would be identical to the case of a and b.

   Since we have analyzed all relevant cases (which produced strings *not* in L), we can conclude by the Pumping Lemma that L is NOT context-free.

   □

4. (20 points) Give a description of a Turing Machine that decides the strings of the form "$u < v$" where u and v are two positive integers in binary and the string is valid as an inequality. Here the alphabet is $\{0, 1, <\}$. (For example $10 < 100$ is in the language but $11 < 10$ is not; Also note that the leftmost digit of the binary representation of every positive integer is 1.) Explain why your Turing Machine works. Your description can have high level lines such as "Scan the tape to the right, until you see the first 0".

   Step 1: Check if the first symbol of u is a "1". If it is not, reject. If it is, cross it off and check if the first symbol of v is a "1". If it is not, reject. If it is, cross it off and go to step 2. This ensures that u and v are positive numbers.

   Step 2: Cross off the first (non crossed-off) symbol in u , remember it as "a", and go to step 3. If there is no such non crossed-off symbol symbol in u, if all of v's symbols are crossed off, then reject. Else, accept. Essentially, what we are doing here is if we see a non crossed-off symbol in u, we still need to compare that symbol with the corresponding symbol in v. If u and v contain no non crossed off symbols, then that means that we have failed to find a digit in v which exceeds the corresponding digit in u, so we reject. Otherwise, v was longer, so we can accept.

   Step 3: Check the first non crossed-off symbol in v. If this is the blank symbol, then reject (u was longer than v). If this symbol is greater than a, then check if u is longer than v (by crossing out one of u_i's and one of v_i's one at a time). If u is longer, than reject (v cannot be greater than u if it is shorter). Otherwise, accept (v had a different corresponding digit at a higher power than any possible other corresponding digits following it). If it is equal to a, then cross it off and go to Step 2 (still need to check other digits). If it is less than a, check if v is longer than u. If so, accept. Otherwise, reject (reasoning symmetrical as before).

   (Note: We use the "<" symbol to distinguish between u and v. u is before the "<" symbol, while v is after the "<" symbol).

5. (20 points) Prove that a PDA that has access to two stacks is as powerful as a Turing Machine. Such a PDA has labels on the transition arrows of the form $(a, \alpha, \beta \to \alpha', \beta')$, meaning that read an a from the input, pop an α from the first stack, β from the second stack, and push α' on the first stack, and β' on the second stack.

   Proof. Let T be some Turing Machine and let P be some PDA. P has two stacks, S_1 and S_2. To prove the equivalence of T and P, we will show that P can simulate all operations of T's tape.

   Let S_1 store all letters $a_1...a_n$ on the left side of the head of the tape such that a_n is at the top of the stack and a_1 is at the bottom. Similarly, let S_2 store all the letters $b_1...b_n$ on the right side of the head of the tape such that b_1 is at the top of the stack and b_n is at the bottom.

Next, we store our current position at the top of S_2.

Read the current symbol: We simply examine the top of S_2.

Move left: To move left to some a_i, we pop all the letters from a_n to a_i from S_1, and we push those letters on S_2.

Move right: To move right to some b_i, we pop the current position and all the letters from b_1 to b_{i-1} from S_2, and we push those letters on S_1.

Edit symbol: To edit a symbol, we move to that symbol (as described above), pop that symbol from S_2, and push a new symbol to S_2.

Since we have shown that P can simulate all operations of T, we have shown that P and T are equivalent.

□