CS 4510: Automata and Complexity

Spring 2015

Home work 3 // Due: Friday, February 19, 2015

Sagar Laud, slaud3, 902792910

1. (15 points)

The iterations are as follows. Naturally the initial contents are the state of the table when k=0.

k=0

	p0	p1	p2	p3	p4	p5
p0			0	0	0	0
p1			0	0	0	0
p2						
p3						
p4						
p5						

k=1

	p0	p1	p2	p3	p4	p5
p0			0	0	0	0
p1			0	0	0	0
p2					1	1
p3					1	1
p4						
p5						

k=2

	p0	p1	p2	p3	p4	p5
p0			0	0	0	0
p1			0	0	0	0
p2					1	1
p3					1	1
p4						
p5						

So we see now that between k=1 and k=2, there was no change. As a result, we are done with our algorithm. We can now combine our states to determine our overall DFA. Looking at the overall results, we can merge p0 and p1 together, merge p2 and p3 together, and merge p4 and p5 together. Let us just assume that the merged states keep the names of the earliest named state. So, our state space shrinks to p0, p2, and p4.

We may formally define a DFA now:

- Q = {p0, p2, p4}
- Σ = {0, 1}
- F = {p0}
- δ is defined as follows:
  - δ(p0, 0) = p0
  - δ(p0, 1) = p2
  - δ(p2, 0) = p2
  - δ(p2, 1) = p4
  - δ(p4, 0) = p4
  - δ(p4, 1) = p0

So, looking at this DFA, it is clear that it accepts the set of strings for which the number of 1's in the string is a multiple of 3.

Sagar Laud, slaud3, 902792910

2. (5 points)

   This problem requires two parts. We see a recursive definition for S which means that for starters, on the outside we will have 0^k and 1^k where $k \geq 0$. The rest of it requires some thinking. At first it looks like $\{0, 1\}^+$ however it is more complex than that. The remainder of the rules for S restrict us somewhat. They force the string to either start with a 1 or end with a 0. This was one key point that we needed to find otherwise we could simply put $\{0, 1\}^+$ in between 0^k1^k. From there the rest is simple. Let us define a string x that is an element of our middle portion. So, our language is 0^kx1^k where x is an element of $\{0, 1\}^+$ such that x must start with a 1 or end with a 0 (it may do both).

Sagar Laud, slaud3, 902792910

3. (10 points) We will need 4 rules to describe this language using a CFG. So for starters, let us define our start variable S. Now, we look at individual cases. For each a that we have, we must have either a c or a d. Likewise, for every b we have, we must also have a c or a d. So we define our first rule as such. S $\rightarrow$ aAc | aBd | bCc | bDd | ϵ. So from here, we must add rules for each of the variables A, B, C, and D. We can substitute more a's and c's in this step if we want or we can switch to b's and c's. So, we define A $\rightarrow$ aAc | bCc | ϵ. The same logic applies to our rule B except that we cannot repeat any a's and c's. So, C $\rightarrow$ bCc | ϵ. Interestingly enough, we can also replace B in the initial rule simply because between a and d, we can have any of these combinations. So we amend the first rule to be S $\rightarrow$ aAc | aSd | bCc | bDd | ϵ. We end by defining our final rule which functions very similarly to rule A except that it handles b's and d's instead of a's and c's. So, D $\rightarrow$ bDd | bCc | ϵ. Those are all we need. So to summarize, the CFG is

   - S $\rightarrow$ aAc | aSd | bCc | bDd | ϵ
   - A $\rightarrow$ aAc | bCc | ϵ
   - C $\rightarrow$ bCc | ϵ
   - D $\rightarrow$ bDd | bCc | ϵ

Sagar Laud, slaud3, 902792910

4. (20 points)

   Proving this with the Myhill-Nerode Theorem relies on the concept of distinguishability. Let us look at our language and choose strings that aid in this proof. Let us define x and y as elements of Σ^* such that x $\neq$ y, |x| = |y|, and x, y $\notin$ L. There are an infinite number of such strings. From here we rely on the concept of distinguishability. Let us now define a string z = x^R. Now, we proceed to append this string to our strings x and y. In doing so, we show that the strings are distinguishable by L. When we concatenate the strings, we see that xz $\in$ L while yz $\notin$ L. In doing this, we have shown that there is an infinite number of equivalence classes. As a result, we have proven that L is non-regular by virtue of the Myhill-Nerode Theorem.

   The context free grammar which describes this language is pretty simple to derive.

   We start with our original rule: S $\rightarrow$ 0X0A | 1X1A. This basically allows us to make one portion of the string a palindrome with the remaining string which is derived from A representing the remaining portion. Our next rule then is to replace X with the same possibilities: X $\rightarrow$ 0X0 | 1X1 |ϵ. Finally, we provide a rule to use for A to allow that final portion to generate the last portion. A $\rightarrow$ 1A | 0A |ϵ. So the final context free grammar is:

   - S $\rightarrow$ 0X0A | 1X1A
   - X $\rightarrow$ 0X0 | 1X1 |ϵ
   - A $\rightarrow$ 1A | 0A |ϵ

Sagar Laud, slaud3, 902792910

5. (10 points)

The logic for this uses the same types of logic that we used in past assignments. Let us define four PDAs A, B, C, and D. $A = (Q_a, \Sigma_a, \Gamma_a, \delta_a, q_a, F_a)$, $B = (Q_b, \Sigma_b, \Gamma_b, \delta_b, q_b, F_b)$, $C = (Q_c, \Sigma_c, \Gamma_c, \delta_c, q_c, F_c)$, $D = (Q_d, \Sigma_d, \Gamma_d, \delta_d, q_d, F_d)$. A accepts the set of strings in our language where #0s < #1s, B accepts the set of strings where #0s = #1s. Then, C accepts the set of strings where #1s < 2*#0s, and D accepts the set of strings where #1s = 2*#0s. So now we combine these machines with epsilon transitions. We can then go ahead and define two machines, X and Y. X simply uses epsilon transitions the same way that we did with our NFAs. Essentially, $\epsilon, \epsilon \to \epsilon$. So we use these transitions to define our machines X and Y such that X accepts if either A or B accept and Y accepts if either C or D accept. We have effectively defined two machines, X will accept all strings for which #0s $\leq$ #1s. Likewise, we have created a separate machine Y which accepts all strings where #1s $\leq$ 2*#1s. Now we just abstract this out one more level. We define our Machine $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$, such that it starts off with epsilon transitions as explained before to X and Y. Our machine M then accepts if X accepts and Y accepts. In doing this, we have effectively designed a PDA that accepts the required language L.

Sagar Laud, slaud3, 902792910

6. (10 points) We know that Chomsky Normal Form has properties that must be enforced. We can use these properties to our advantage. Because the string is guaranteed to be of length greater than zero, we can continue. Each rule which maps a variable to variable(s) can be substituted, starting with our start variable. Each one of these steps will add in a new non-terminal. There will be n-1 of these steps in total. From here, we must use rules that map to terminals. As our string is of length n, we must take n steps to do this. As a result, we take n-1 steps to set up our variables and then n steps to map them to the corresponding terminals. As a result, this will take n-1 + n = 2n-1 steps.