CS 4510: Automata and Complexity

Spring 2015

Home work 1 // Due: Friday, January 16, 2015

Sagar Laud, slaud3, 902792910

1. (15 points) For $k \geq 1$ and $p \geq 2$, let

$$L_{k,p} = \{w \in \{0, 1, \cdots, p - 1\}^*\} \mid w \text{ is a } p\text{−ary representation of a multiple of } k\}.$$

We presented a DFA for $L_{5,3}$ in class. Generalize the construction for arbitrary k and p. Give arguments to show that your construction is correct.

For this solution, we rely heavily on the example presented in class.

- Let us begin with the definition of D(w). For any p-ary string, let D(w) represent it's decimal value.
- Let us also assume that the empty string is in our language so that D(ϵ) = 0
- Let us now take a word from {0, 1, 2, ... p-1}*. We can then make the claim that D(w) mod k = i. Specifically, we can now say that D(w) = k * q + i for q, i being elements of {0, 1, 2...k-1}
  - We now consider the case where we have D(wj), where j is an element of {0, 1, 2...p-1}.
  - We see that D(wj) = k * D(w) + j = k * (p * q + i) + j
  - Consequently, we note that D(wj) mod k = (p * i + j) mod k
- After these derivations, we can now define our DFA X
  - X will have k states q_0 to q_{k-1}
  - q_0 will be our start state as well as the only accepting state
  - $\hat{\delta}(q_0, w) = q_{D(w) mod k}$
  - Therefore, we define our transition function as: $\delta(q_i, j) = q_{(p*i+j) mod k}$
- Finally, we prove our construction's validity with a proof by induction
  - We claim that $\hat{\delta}(q_0, w) = q_{D(w) mod k}$
  - The base case is of course that when $|w| = 0$, this is indeed True
  - Now for $|w|$ which are greater than or equal to 1:
    * Let w = yj, with y as an element of {0, 1, 2, ... p-1}*, j as an element of {0, 1, 2, ... p-1}
    * So from here, $\hat{\delta}(q_0, yj) = \delta(\hat{\delta}(q_0, y), j) = \delta(q_{D(y) mod k}, j) = q_{(p*D(y)+j) mod k} = q_{D(yj) mod k}$

Sagar Laud, slaud3, 902792910

2. (15 points) For a fixed k, let $L_k = \{w \in \{0,1\}^* \mid \text{the } k-\text{th symbol from the right is } 1\}$. Design a DFA for L_k.

   For this solution, we rely heavily on the example presented in class. We will begin by storing the last k bits in the states of the DFA. In this way, our DFA will have 2^k states, which is indeed finite. We will then define our transition function as such: for each state labelled $x_1x_2x_3...x_k$ (for each x_i as an element of $\{0, 1\}$), we will transition from our current state to $x_2x_3...x_k0$ upon reading a 0 and to $x_2x_3...x_k1$ upon reading a 1. Our accept states then are the states labelled $1x_2x_3...x_{k-1}$, for each x_i as an element of $\{0, 1\}$.

Sagar Laud, slaud3, 902792910

3. (15 points) Let A and B be two regular languages over $\Sigma = \{0, 1\}$.

   Show that d$isjunct(A, B) = \{x \vee y \mid x \in A, y \in B, |x| = |y|\}$ is also regular. Here $x \vee y$ is the Boolean OR of the two bit strings x and y.

   We will use the magic properties of an NFA to show that this statement is true. Let us start by defining DFAs for A and B. As these languages are regular, by definition, there must be DFAs that recognize them individually. Let us define DFAs X and Y such that X recognizes A and Y recognizes B. We will formally define $X = (Q_1, \{0, 1\}, \delta_1, q_1, F_1)$. Likewise, we will define $Y = (Q_2, \{0, 1\}, \delta_2, q_2, F_2)$. We can now use these to our advantage. So now, if our current input symbol w_i is a 1, then our current bits (x_i, y_i) must be one of three things. Either (1, 0), (0, 1), or (1, 1). Likewise if our current input symbol is a 0, then our current bits must be (0, 0). Now, we have everything that we need to design our NFA. The solution is to design an NFA, $Z = (Q, \{0, 1\}, \delta, q_0, F)$.

   - $Q = Q_1 \times Q_2$
   - For all x that are an element of Q_1 and y that are an element of Q_2, we define the following transition function.
     - $\delta((x, y), 1) = \{(\delta_1((x, y), 1), \delta_2((x, y), 0)), (\delta_1((x, y), 0), \delta_2((x, y), 1)), (\delta_1((x, y), 1), \delta_2((x, y), 1))\}$
     - $\delta((x, y), 0) = \{(\delta_1((x, y), 0), \delta_2((x, y), 0))\}$
   - $q_0 = (q_1, q_2)$
   - $F = F_1 \times F_2$

Sagar Laud, slaud3, 902792910

4. (15 points) Let A and B be two regular languages over a finite alphabet Σ.

Define $shuffle(A, B) = \{w|\ w = a_1b_1 \cdots a_kb_k,$ where each $a_i, b_i \in \Sigma^*, a_1 \cdots a_k \in A$ and $b_1 \cdots b_k \in B\}$.

Show that $shuffle(A, B)$ is regular.

The simplest way to do this is to construct an NFA which relies on two DFAs. Let us first define the DFAs M_A and M_B. $A = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and $B = (Q_2, \Sigma, \delta_2, q_2, F_2)$. Let M_A recognize A and let M_B recognize B. From here, we can proceed to construct an NFA X as follows. Let us define $X = (Q, \Sigma, \delta, q_0, F)$ such that:

- $Q = Q_1 \times Q_2$
- The transition function is the crucial part to this. In order to transition, we either want to transition in Machine A and leave Machine B alone or vice versa, this too will be done nondeterministically.
  Therefore, with x being the state in M_A, y being the state in M_B, and z being an element of Σ, the transition function is:
  - $\delta((x, y), z)) = (\delta_1(x, z), y), \delta_2(x, (y, z))$
- $q_0 = (q_1, q_2)$
- $F = F_1 \times F_2$

Sagar Laud, slaud3, 902792910

5. Let L be an infinite subset of $\{1\}^*$. Show that L^* is regular.

   This solution is rather interesting. Let us start with a string 1^k that is an element of L. Then we can extrapolate this a little to use another arbitrary string 1^r that is an element of L*, where $r = ak + b$, b is an element of {0, 1, 2...k-1}. So now if we examine the strings $1^{r+k}, 1^{r+2k}$... we see that each of these strings are in L*. In fact, we see that we can continue to add an infinite number of concatenations of 0 to k 1's in order to continue to extend our string. Thus, we have created a way to create strings in L*. We can continue this till infinity if we wish and depending on the values that are chosen for k and b, we can create any string in L*. Now we can think about the modulus operator. By using our integer k as our baseline and using multiples of that plus a remainder, we can take any string from L* and partition it into sets that are regular. Basically, we can choose a value for k and with any string in L*, we can partition it into x number of sets of 1^k plus a concatenation of anywhere from 0 to k-1 1's. In this way, we have shown that any string from L* can be partitioned into sets that are regular, therefore making L* regular.