CS 4510: Automata and Complexity

Spring 2015

Home work 6 // Due: Friday, April 17, 2015

Sagar Laud, slaud3, 902792910

1. (15 points) Show that any non-trivial property $\mathcal{P}$ of recognizable sets that includes the empty-set $\emptyset$ is not recognizable.

   (Hint: Give an alternative proof of Rice's theorem that uses a reduction from $\overline{A_{\text{TM}}}$ to any non-trivial property $\mathcal{P}$ that includes TMs whose language is $\emptyset$.)

   Let us start with what we know. Because P is a non-trivial property, each Turing Machine whose language is $\emptyset$ is an element of P. However, because the property is non-trivial, there exists a machine such that the language of that machine is not an element of P. Let us take a machine $M \in P$ and $N \notin P$. Finally, let us define X as a TM which will reduce $\bar{A_{TM}}$ to P as follows:

   On a given input tuple consisting of a machine M and a string w, X will construct a TM for us. Let us refer to this TM as K. Upon the input, the following will happen.

   When N receives an input x, it will run the string w through the machine M. If M accepts w, then run N on x. If N accepts our input x, then accept. Else if N rejects x, then reject. Finally, if M does not accept w to begin with and halts and rejects w, then we reject. By constructing this, we have reduced a non-recognizable problem to another problem. Let us show what the cases are to make this concrete.

   If our inputs M and w are elements of $\bar{A_{TM}}$ then M will either reject w or halt. As a result, the language of K will be the empty set thereby making K an element of P. If the opposite happens and M and w are not elements of $\bar{A_{TM}}$, then it is clear that M will accept w. As a result, the languages of K and N will be equivalent thereby making $K \notin P$. This proves the second case as $K \notin P$ as $N \notin P$.

2. (15 points) The *Constrained* PCP problem is the PCP problem in which each tile can be used in a match at most once.

   That is, the Constrained PCP problem is the following:

   $$\{\left[\frac{t_1}{b_1}\right], \left[\frac{t_2}{b_2}\right], \ldots, \left[\frac{t_k}{b_k}\right] \mid \exists \text{ DISTINCT indices } (i_1, i_2, \ldots, i_m) \text{ such that } t_{i_1} t_{i_2} \ldots t_{i_m} = b_{i_1} b_{i_2} \ldots b_{i_m}\}.$$

   where $t_1, t_2, \ldots, t_k, b_1, b_2, \ldots, b_k$ are strings over some alphabet Σ.

   Is the Constrained PCP problem decidable? Prove your answer.

   The Constrained PCP Problem is decidable. The number of possibilities that can occur without repetition is finite though it may be very large. Meaning, if we have k dominos, since we can choose to omit dominos, the overall number of possibilities is (assuming we require atleast one domino to be present) $\sum_{i=1}^{k} \binom{k}{i} i!$. This is a large number but it is finite. The issue which makes PCP undecidable is the fact that there are an infinite number of possibilities. In this case however, the number of possibilities we have is finite. As a result, the problem would take time to solve but it is indeed decidable since we can try every possibility. A decider would try every possibility and accept if any combination yields a match, and will reject if it exhausts all of the options without a match.

3. (10 points) Problem 7.41, page 327 of the text.

   In the proof of Cook-Levin theorem, a window is a 2×3 rectangle of cells. Show why the proof would have failed if we had used a 2×2 window.

   The issue arises specifically when the head of the Turing Machine moves left. If we use a 2x3 window, then we can have a situation where the top row represents a valid configuration while the second row represents a configuration which could not be produced from the top configuration via the transition function. However, with 2x2 windows, all configurations are valid causing a problem.

   Let us consider a concrete example. Let us define a 2x3 window where the first row is 0 q0 0 and the second row is q1 0 q2. The first configuration is clearly possible and the second configuration is just impossible. This is something that a 2x3 window can catch. However let us look at this in terms of 2x2 windows. If this happens, then we have two cases to address. The first case is if the first row is 0 q0 and the second row is q1 0. This could be a valid transition in reality. However, because the window can't see if the head is reading a 0 or a 1, it just assumes that it's valid. A possible transition that could allow this to occur is $\delta(q_0, 1) = (q_1, 1, L)$. The other case is if the first row is q0 0 and the second row is 0 q2. This too could occur from a valid transition such as $\delta(q_0, 0) = (q_2, 0, R)$. However, if both these checks pass, then we have an issue and the proof is no longer valid.

4. (15 points) Problem 7.43, page 327 from Sipser text.

   For a CNF formula with n variables and m clauses, show that you can construct in polynomial time an NFA with $O(nm)$ states that accepts all non-satisfying assignments, represented by Boolean strings of length n. Conclude that the problem of minimizing NFAs cannot be done in polynomial time unless $\mathcal{P} = \mathcal{NP}$.

   Let us construct the NFA for this purpose to show that if it were to be done, then P would equal NP. Let us call this NFA M. When N receives the formula as an input, it will have m epsilon transitions to each of the clauses. From there, it will simply read the n variables. If the clause is not satisified or the length of the clause is not equal to n, it will accept and will reject if the clause is satisfied. Since we have m clauses, each of which require n states, M has O(mn) states. This algorithm itself can be run in polynomial time.

   Proof of correctness for construction: We have to prove two directions. If the NFA is constructed as above, then it will accept iff there exists a non-satisfying assignment for atleast one clause in the machine. The first direction is simple, for any non-satisfying assignment which we use, atleast one clause is not satisfied and therefore the machine will accept. Going the other way, if the machine accepts an assignment, then atleast one clause is not satisified which makes this assignment non-satisfying. As we have proven both directions, we have shown that this is correct.

   Now we will use a reduction to show that the minimization algorithm cannot be performed in polynomial time unless P = NP. Let us assume that the problem can be solved in polynomial time. Let us pass our algorithm above a problem that is an instance of the 3CNF-SAT problem. So, let our algorithm take in this CNF formula with m clauses and construct the corresponding NFA which accepts all non-satisfying assignments of the formula. Looking at how the problem has been setup, if the CNF formula which we have received is non-satisfyable, then our NFA will actually accept Σ^* assuming we have a binary alphabet. So, reduce the NFA. Upon doing so, if it reduces to a single state and accepts Σ^*, then reject. Otherwise, accept. By doing this, we have shown a polynomial time solution for 3CNF-SAT. This is only possible if P = NP.

5. Let $G = (V, E)$ be an undirected graph. Let C be a collection of pairs of edges of G. A matching M in G is said to *respect* C if at most one edge from each pair in C is in M.

   The *Matching with Forbidden Edges* problem is defined as follows:

   INPUT: An undirected graph $G = (V, E)$, a collection C of pairs of edges of G,and an integer $1 \leq k \leq |E|$.

   PROPERTY: G has a matching M of size k that respects C.

   The *Independent Set* problem is defined as follows:

   INPUT: An undirected graph $G = (V, E)$, and an integer $1 \leq k \leq |V|$.

   PROPERTY: G has an independent set of size k.

   (15 points) Give a polynomial time mapping reduction from the *Independent Set* problem to the *Matching with Forbidden Edges* problem.

   Theoretically we can do this by taking the complement of our graph G. By this I mean, if our graph has an independent set of size k, then we can easily find a set of edges such that at most one edge from each pair in C is in our graph. In fact, none of the edges in C will be in our graph at all. If we have an independent set of size k in G, all we have to do is take the complement of G. This will create a graph with k choose 2 edges in it, none of which are in G and therefore, none of which are in C at all. Since the number of edges is larger than k, we can choose any subset of edges of size k and proclaim it to be the answer. So we have shown a polynomial time mapping. Taking the complement is polynomial time and choosing a subset of size k is of polynomial time as well. As a result, because we have shown a generic way to convert an instance of the independent sets problem to the matching forbidden edges problem, we have completed the mapping reduction.

6. Given two $n \times n$ matrices A and B with 0/1 entries, two $n \times 1$ vectors X and Y with 0/1 entries, and a natural number k, the problem is to decide if the maximum value in the vector $A \times X + B \times Y$ is at most k. (Here $A \times X$ is the product of the matrix A with the vector X. Similarly, $B \times Y$ is the product of the matrix B with the vector Y.)

   (10 points) Show that this problem is in $\mathcal{L}$.

   (See section 8.4, pages 320-321 of the Sipser text for the definition of the class $\mathcal{L}$.)

   To do this, we have to store as few numbers as possible during our computations. Essentially, we want to find the maximum value that results from A * X and the maximum value which results from B * Y. We want to add them together and then subtract k from the result and see if the result is greater than or equal to 0. We're in luck because addition, subtraction, and multiplication can be done in log(n) space. We will need 3 cells for the multiplication of A * X and B * Y. Begin by multiplying values bit by bit and storing them in the first cell. Sum up all the numbers whose sum represents the first of n cells (this is the nx1 matrix which represents the solution). For each iteration, each change in the value of n for the nx1 matrix, store the result in a second cell. Calculate the difference between the two cells. If the difference is positive, then do not update the first cell. If it is negative, then update the first cell as a new maximum has been found. Repeat this process until the maximum value from A * X has been found. Then, do the same with B * Y (we will only need one more cell, we can just shift our machine over one cell and save the value of A * X in the first cell). After getting the maximum value of B * Y, sum the two numbers and put them in the third cell. We can do this simply by adding the bits until the result sits in the third cell. Finally, subtract k from the sum of the two numbers. If the difference is greater than zero, reject. Otherwise, accept.