COMP 330 Assignment 6 Solutions and Grading Guide

Prakash Panangaden

Question 1[20 points] For each of the following assertions give *brief* arguments indicating whether they are true of false. In each case I am talking about sets of positive integers.

a. For each $n \in \mathbb{N}$ we have a computable set C_n. The set $\bigcup_n C_n$ is computable. We assume that the collection of computable sets is *effectively given*: this means that there is an algorithm that reads a natural number n as input and outputs a description of a Turing machine that decides the set C_n. **Solution**: This is **false**. Consider the sets $A_n = \{\langle M, w\rangle | M \text{ accepts } w \text{ in } n \text{ steps}\}$. Each of these is computable, as we need only run M on w for n steps. However, their union $\bigcup_{i=0}^{\infty} A_i$ is in fact exactly the set $A_{TM} = \{\langle M, w\rangle | M \text{ accepts } w\}$ which is not computable.

b. For each $n \in \mathbb{N}$ we have a computably enumerable set C_n effectively given as described above. The set $\bigcup_n C_n$ is computably enumerable. **Solution**: This is **true**. Suppose that we have a family of semi-decision procedures A_i for the languages L_i. Given a word w we ask each of the A_i whether they accept w. Since they are only semi-decision procedures we must dovetail the computations. If w is in the union, then one of the A_i will have to accept it eventually. If w is not in the union we may never find out. Thus, the union is CE.

There is quite a lot of confusion about the first question. If they say that every singleton is computable thus every set is the union of computable sets they should lose 5 marks because the union in that case is not effectively given. If they argue that the union is CE that is not ideal but they should only lose 2 marks. For full marks they have to give an example. For the second part they should get it. Of course, drivel gets 0.

Question 2[25 points] For this question the alphabet is $\{a, b\}$. Suppose that the language L is CE but not computable; this means that $\overline{L}$ cannot be CE. We define a new language as follows:

$$K = \{aw | w \in L\} \bigcup \{bv | v \in \overline{L}\}.$$

1. Show that K is not computable. [5 points]
2. Show that K is not CE. [10 points]
3. Show that K is not co-CE. [10 points]

Solution: Here is a beautifully-written elegant version of the solution due to Kimberly Trickey, one of the many excellent students I had in the class of 2014:

1. Define $f : \Sigma^* \to \Sigma^*$ by $f(w) = aw$. Clearly, f is total computable. From the definition of K we have
$$x \in L \iff f(x) \in K,$$
thus we have a mapping reduction $L \leq_m K$. Since L is not computable K cannot be computable.

2. We define the function $g(w) = bw$; again, this is obviously total computable. We have $x \in \overline{L} \iff g(x) \in K$ so we have shown another mapping reduction $\overline{L} \leq_m K$. Since L is CE but not computable we have that $\overline{L}$ is not CE so K cannot be CE.

3. Since L is CE and not computable it is **not** coCE. Thus the mapping reduction in part (1) shows that K is not coCE.

People in general will not give such nice answers. However, take points off only if they have said something really wrong. They are still struggling to express themselves.

Question 3[15 points] Suppose that M is a Turing machine and w is a word. Is the question "Does M ever use more than 330 cells on its tape while processing w?" decidable or not. Prove your answer.

Solution This is decidable for a given input. For simplicity, we'll assume the machine is deterministic. The argument can work just as well if not. Let $M = (Q, \Sigma, \Gamma, \delta, q_0)$ be the machine in question, and w the given input. The key is to note that there are only finitely many configurations in which M can be, such that it has not gone beyond the 330th cell on its tape. In fact, there are $N = 330 \cdot |Q| \cdot |\Gamma|^{330}$ such configurations: 330 positions the head can be in, $|Q|$ states it could be in, and $|\Gamma|^{330}$ possibilities for the contents of the 330 cells.

If the machine never goes beyond 330 cells, then by the pigeonhole principle some configuration must be repeated in the first N steps. As the machine is deterministic, once it reaches a configuration it reached before, it will loop, but never using more than 330 cells. On the other hand, if it will go beyond 330 cells, it must do this in no fewer than $N + 1$ steps. So to decide this, run the machine for N steps. If a configuration is repeated, the machine never uses more than 330 cells on its tape; if not, then it will.

For guessing incorrectly they should get half the marks off. If they guess incorrectly and then say it is because of Rice's theorem they should get zero. I specifically told them that Rice's theorem will not help them. They need to state clearly that they are looking for a loop. If they are unclear about this they should lose 5 points. If they guess correctly and give no reason they should lose 10 points.

Question 4[20 points]

1. Here is a question on regular languages just to get you in shape for the final exam. Suppose that L is a regular language and w is any word, not necessarily in L. We define the set
$$L/w = \{x \in \Sigma^* | xw \in L\}.$$
Show that L/w is regular. [5 points]

   Solution: Suppose that $M = (Q, \Sigma, q_0, \delta, F)$ is a DFA for accepting L. We define a new DFA $M' = (Q, \Sigma, q_0, \delta, F')$. This NFA has the same states, the same start state, the same transition function and the same alphabet as M. The final states are different. We define the set of final states as follows:

$$F' = \{q \in Q | \delta^*(q, w) \in F\}.$$

Of course, it could happen that the set F' turns out to be empty in which case L/w would be the empty language.

2. Suppose that G is a context-free grammar. Show that the question "Is $L(G)$ regular?" is undecidable. Here is a possible approach. Let N be some language that is known to be context-free but not regular (for example, $\{a^n b^n | n \geq 0\}$). Now consider the language $L = N\#\Sigma^* \cup \Sigma^*\#L(G)$, where $\#$ is some symbol that is not in $L(G)$ or N. Prove that L is always context-free but is regular if *and only if* $L(G) = \Sigma^*$. This, by itself, does not complete the question, so you have to complete all the remaining steps as well as proving the claim. Also, you should think about why I put part (1) together with this question? You are free to ignore this hint, but if you do so, I will mark you just as rigourously as the people who used the hint. [15 points]

   Solution: This is a reduction argument. We are going to use the fact that it is undecidable for a CFG whether its language is Σ^*. The hint tells us how to cook up a CFL with the required properties. Since N is context-free we have that $N\#\Sigma^*$ is context-free and clearly also $\Sigma^*\#L(G)$ is context-free. Since the union of two context-free languages is context-free, the language L is context free. Now if $L(G) = \Sigma^*$ the language L is just $\Sigma^*\#\Sigma^*$ which is clearly regular. Suppose that $L(G) \neq \Sigma^*$, then there is a word w in Σ^* that is not in $L(G)$. Now consider the language $L/\#w$. Since $w \notin L(G)$ this is just N. Now we choose N so as to be not regular thus, in this case, $L/\#w$ is not regular and hence L is not regular. Thus we have shown that L is regular iff $L(G) = \Sigma^*$. If we could decide whether a CFG produces a regular language we could use the above transformation and find out for any CFG(G) whether $L(G) = \Sigma^*$, but this is known to be undecidable.

   Please mark this carefully. See that they have the reduction required and not some garbage. If they have not given a proper reduction then they should lose at least half the marks (i.e. 10). If they show that they have not even understood what we are trying to do give them zero.

Question 5[20 points] You are playing a video game under the following idealized conditions. The computer memory is unbounded and you have no time limit for finishing the game. The game board is the set of points in the plane with integer coordinates and time moves in discrete integer steps. There is a hidden submarine: you do not not know its location, you do not know its speed and you do not know its direction of motion. The speed and direction of motion do not change throughout the game. The speed is a natural number and the direction of motion is either "up", "down", "left" or "right". For example, the submarine could start at $(2, 3)$ have speed 7 and move right. Then at step 0 it is at $(2, 3)$, at step 1 it is at $(9, 3)$, at step 2 it is at $(16, 3)$ and so on. At every step you get to zap a point: you enter the coordinates and if the submarine is at that point, *at that time step*, you will destroy it. Of course, there is no point zapping a position before it gets there or after it leaves. Give a *strategy* or *scheme* that is *guaranteed* to get the submarine *at some finite stage*. I repeat, you do not know where it started, you do not not know its direction and you do not know its speed; you only know that the speed and direction do not change. You get zero points for this question if your strategy works probabilistically. Hint: Ask yourself, "why is this on the homework for this class?"

Solution: There are 4 parameters that determine the motion of the submarine completely: the x and y coordinates of the starting point, the speed and the direction. The first three are integers and the last one can take on one of 4 values. At every stage I make a guess of what these parameters are and I compute where it would be now with that guess. If I am at stage n and I guess that it starts at (x_0, y_0) with speed s and direction right then I can compute its present position is $(x_0 + n * s, y_0)$. Similarly, if I guess that the direction is up the present position must be $(x_0, y_0 + n * s)$. Now all we have to do is make sure that we try every single possible combination systematically. The function τ encodes pairs of integers as a single integer and can be used to encode arbitrary finite tuples of integers as integers. (Actually we encoded pairs of natural numbers as a natural number but it is easy to modify it slightly to encode tuples of integers as natural numbers.) Thus we look at the encoding of each guess as a natural number and at each step n we view n as the encoding of a guess, decode n to get the parameters of the guess, do the simple computation to determine the present position of the submarine and zap the resulting point. Since we are systematically trying all possible guesses we are sure to hit the submarine eventually. Note this works as long as the submarine moves according to an arbitrary total computable function even in an N-dimensional grid!

They must clearly explain how the scheme works. If they just say "dovetailing" take 10 points off for not being explicit enough.