Homework 6. Due December 4th, 11:59AM.

CS181: Fall 2021

GUIDELINES:

- Upload your assignments to Gradescope by 9:59 PM.
- Follow the instructions mentioned on the course webpage for uploading to Gradescope very carefully (including starting each problem on a new page and matching the pages with the assignments); this makes it easy and smooth for everyone. As the guidelines are simple enough, bad uploads will not be graded.
- You may use results proved in class without proofs as long as you state them clearly.
- Most importantly, make sure you adhere to the policies for academic honesty set out on the course webpage. The policies will be enforced strictly. Homework is a stepping stone for exams; keep in mind that reasonable partial credit will be awarded and trying the problems will help you a lot for the exams.
- Note that we have a **modified grading scheme for this assignment**: A sincere attempt will get you 100% of the credit and a reasonable attempt will get you 50% for each problem. Nevertheless, please attempt the problems honestly and write down the solutions the best way you can - this is really the most helpful way to flex your neurons in preparation for the exam.
- All problem numbers correspond to our text 'Introduction to Theory of Computation' by Boaz Barak. So, exercise a.b refers to Chapter a, exercise b.

1. Give a quantified integer statement to express the following: [2 points]

   (a) "There are an infinite number of primes". [Hint: First design a logical statement (that only uses arithmetic operations and other quantified integer variables) with one free variable p, $Prime(p)$ that is true only when p is a prime. You can then combine this expression with an idea similar to examples we saw in class.]
   Solution. Define the formula $Prime(p) = \forall i, j, (NOT(i \cdot j = p) \vee (i = 1) \vee (j = 1))$. This formula just says that if $i \cdot j = p$, then either $i = 1$or $j = 1$. (Recall that $f \Rightarrow g$ is logically equivalent to $NOT(f) \vee g$. The statement there are infinite number of primes can be written as

   $$\forall m, \exists p, (p > n) \wedge Prime(p).$$

   Expanded out, the above expression is:

   $$\forall m, \exists p, (p > n) \wedge (\forall i, j, ((NOT(i \cdot j = p) \vee (i = 1) \vee (j = 1)))).$$

(b) "1729 is the smallest natural number that can be expressed as sum of two cubes in two different ways".

Solution. Let us find a statement that $P(n)$ that is true only if n can be expressed as sum of two cubes in two different ways. Note that we have to rule out the two ways being just a rearrangement of one such way. We can take:

$$P(n) \equiv \exists a, b, c, d(n = a \times a \times a + b \times b \times b) \wedge (n = c \times c \times c + d \times d \times d) \wedge \neg(a = c) \wedge \neg(a = d).$$

Now, you can write down the desired statement as follows:

$$P(1729) \wedge (\forall n((n < 1729) \Rightarrow \neg P(n))) .$$

[The number 1729 is called a *Taxicab* number based on a nice anecdote about the above *fact* and how the famous mathematicial Ramanujan observed it in a casual conversation with G.H. Hardy from a hospital bed.]

2. Exercise 11.2, part (a). [2 points]

[Hint: Try to find a reduction from any of the problems in class. For example, you can try to reduce from NOTEMPTY. Can you find a reduction R such that for any Turing machine M, there exists an input w such that $M(w) = 1$ if and only if $Findproof(R(M)) = 1$?]

Solution. We will show a reduction from NOTEMPTY. Recall its definition: for a TM M, $NOTEMPTY(M) = 1$ if there exists an input x such that $M(x) = 1$ and 0 otherwise. We showed in class that NOTEMPTY is uncomputable (follows from Rice's theorem - NOTEMPTY is a non-trivial semantic property).

Now, let us reduce NOTEMPTY to FINDPROOF. For a machine M, define a new TM V as follows: $V(x, w)$ - (a) Return $EVAL(M, w)$. That is, V ignores the first input and just runs M on w. Now, set $\mathcal{R}(M) = (V, 0)$. Now, it is easy to check that $NOTEMPTY(M) = FINDPROOF(\mathcal{R}(M))$. If there exists a w such that $M(w) = 1$, then $V(0, w) = 1$ so that $FINDPROOF(V, 0) = 1$. Else, both are 0.

This proves that FINDPROOF is uncomputable as NOTEMPTY is uncomputable.

Additional Practice Problems

1. Are the following languages/functions context free? If yes, write a context free grammar. If not, prove it with context free pumping lemma.

   (a) $A = \{x \in A | \#(a,x) = \#(b,x) = \#(c,x)\}$
   Note: $\#(a,x)$ represents the number of character a that exist in word x.
   Example: $\#(a, ababa) = 3$

   Solution: Assume language A is context free, then the context free pumping lemma must hold. Any string x of at least length p that exists in A can be broken down into $x = uvxyz$ where (1) for every $i \geq 0, uv^ixy^iz \in A$ (2)$|vy| > 0$ (3)$|vxy| \leq p$
   Consider string $s = a^pb^pc^p$ We can break up this string s as $s = uvxyz$ such that it satisfies all three conditions.
   Lets pump using $i = 0$, $s' = uv^0xy^0z$. By property (1), we know that s' is also in language A. By pumping down, we have removed some characters in some substring of length at most p. Since the substring is at most length p, we know that it cannot contain a's, b's and c's. The count of one of them must remain unchanged. But if we delete characters from one or two of them without deleting the third, we know that our equality will not hold.
   $\#(unaffected, s') = n$
   $n - |vy| \leq \#(affected, s') < n$

   Since our equality doesn't hold, s' is not in language A. This is a contradiction of (1). The pumping lemma doesn't hold so A must not be context free.

   (b) $L = \{0^{n!} : n > 0\}$
   Solution: Assume the language is context-free. Then by the pumping lemma there exists a number p such that the conditions of the pumping lemma hold. Consider the string $x = 0^{(p+1)!}$. Now, for any way to split up $x = abcde$ such that the three conditons of the lemma hold. Then, ab^2cd^2e must be in the language. However, the length of ab^2cd^2 is at most $(p+1)! + p$ (as the legnth of bcd is at most p). However $(p+1)! + p < (p+2)!$ the next smallest length string in L (when compared to length of x). So ab^2cd^2 is not in the language leading to a contradiction. So L is not context-free.

   (c) $F : \{0,1\}^* \to \{0,1\}$ where $F(x) = 1$ if and only if $x = 0^i1^j$ for some $j = i^2$.
   Solution: Assume the language is context-free. Then, by the pumping lemma for context-free grammars, there exists a number $p$such that the codnitions of the lemma hold. Consider the string $x = 0^p1^{p^2} \in L$. Now, we consider various cases to rule out being able to split $x = abcde$ such that the conditions of the lemma hold.
   Case 1: Suppose b or d involves a 0 and a 1. In this case, the string $x' = ab^2cd^2e$ would have a 1 appearing before a 0 and hence cannot be in the language.
   Case 2: Suppose b or d only involve only 0's. In this case, the string $x' = ab^2cd^2e$ would have more 0's than p but the number of 1's would be the same leading to a contradiction to it being in the language.
   Case 3: Suppose b or d only involve only 1's. In this case, the string $x' = ab^2cd^2e$ would have more than 1's than p^2 but the number of 0's would be just p leading to a contradiction to it being in the language.

Case 4: Suppose b involved only 0's and d involved only 1's. Then, the number of 0's in $x' = ab^2cd^2e$ would be at least $p+1$ whereas the number of 1's is at most $p^2+p < (p+1)^2$ (as bd is of length at most p). Therefore, x' once again cannot be in the language.

In all of the cases, we get a contradiction so the language L cannot be context-free.

2. Write a QIS for the statement: "There are an infinite number of twin primes, i.e., an infinite number of integers p such that p and $p+2$ are both prime.

   Solution: Let $Prime(p)$ be the predicate which is true only when p is a prime (you must have used it in problem 1.(a)). Then, we can design for the *Twin prime conjecture* as the following: $\forall n, \exists p, (p > n) \wedge Prime(p) \wedge Prime(p+2)$.

3. Recall that in class we defined a verifier as a two input algorithm and defined properties of verifiers: *effective*, *sound*.

   Now, call a verifier V *consistent* proof system for QIS if you cannot prove a formula and its negation in F. That is, $\forall F, \neg(\exists s_1, s_2\ (V(F, s_1) = 1) \wedge (V(NOT(F), s_2) = 1)$.

   Show that for every effective verifier V, there exists a QIS $\mathcal{C}(V)$ that is true if and only if V is *consistent*.

   [Hint: Recall that as part of our statement that $\mathcal{U}_{QIS}$ was uncomputable, we claimed that there is a reduction from HALTONZERO. That is for every machine M, there exists a QIS $\mathcal{R}(M)$ that is true if and only if M halts on 0. You can appeal to this reduction directly by designing a suitable TM (no need to know the reduction at all, just use that it exists). You can use high-level programming constructs here.]

   Solution: The idea is to show that there is a machine which will halt only when V is not consistent. Consider the following program:

   def ContradictionChecker:

   (a) For each triple $(F, s_1, s_2) \in \{0,1\}^*$:
      i. If $(V(F, s_1) == 1) \wedge (V(NOT(F), s_2) == 1)$:
      ii. Return NotConsistent.

   We can enumerate over triples by using encoding tricks as we saw early in class or taking a single string in $z \in \{0,1\}^*$ and considering all possible of ways breaking it up as $F \circ s_1 \circ s_2$.

   Note that the above program will halt only if V is not consistent. As if it is consistent, there will be F, s_1, s_2 satisfying the If condition and we return. And it won't halt if V is consistent. So, $HALTONZERO(ContradictionChecker) = 1$ if and only if V is not consistent. Let $\Phi = \mathcal{R}(ContradictionChecker)$. Then, Φ is a QIS such that $\Phi \in \mathcal{U}$ if and only if V is not consistent. We can now take $\mathcal{C}(V) = NOT(\Phi)$ (negation of a QIS is also a QIS).

4. We can use KRT to find *recurisve TMs* in the following sense. Consider a TM $T : \{0,1\}^* \times \{0,1\}^*$ defined as follows.

   On input $(< M >, w)$:

   (a) If $w = 0$ (as an integer), return 1.

   (b) Else, run M on $w-1$ (we are viewing w as an integer), if it returns N, return $N \cdot w$.

Now, if we apply KRT to T, we get a program K such that for all w $K(w) = T(< K >, w)$. What is K computing?

Solution: The K above computes the factorial function! It is essentially equivalent to implement recursive implementation of the factorial function as a TM. One can use KRT (and its explicit construction of K to find self-recursive functions without recursion supported directly). It is quite interesting to play with various T and see what KRT implies for that T.

5. **(Advanced problem)** Use Kleene's recursion theorem to prove Rice's theorem from first principles (i.e., without recourse to Toddler or reductions from other uncomputable problems).

   Solution: Suppose $F : \{0,1\}^* \to \{0,1\}$ is a non-trivial semantic function. For the sake of contradiction suppose that F is computable and is computed by a Turing machine G.As F is non-trivial, there exist TMs A_0, A_1 such that $F(A_0) = 0$ and $F(A_1) = 1$. Now, consider the following Turing machine $T : \{0,1\}^* \times \{0,1\}^*\{0,1\}^*$. On input $(< M >, w)$ do the following:

   (a) Compute $F(< M >)$ by running G on $< M >$ (we can do this by our assumption that F is computable).

   (b) If $F(< M >) = 0$: Run A_1 on w (i.e., execute $Eval(A_1, w)$).

   (c) If $F(< M >) = 1$: Run A_0 on w (i.e., execute $Eval(A_0, w)$).

   In otherwords, T when given $(< M >, w)$ as input first checks if $F(< M >)$ is 0 or 1 and then simulates A_1 on w or A_0 on w.

   So far everything is fine and nothing seems contradictory. However, by KRT there exists a machine K such that for all w $K(w) = T(< K >, w)$.

   What should $F(< K >)$ be? Herein lies the contradiction.

   Case 1: $F(< K >) = 1$. In this case, for all w, $K(w) = T(< K >, w) = A_0(w)$ (from our choice of T). That is K is equivalent to A_0. But then, as F is semantic, $F(< K >)$ should be $F(A_0) = 0$ - a contradiction.

   Case 2: If $F(< K >) = 0$. In this case, for all w, $K(w) = T(< K >, w) = A_1(w)$ (from our choice of T). That is K is equivalent to A_1. But then, as F is semantic, $F(< K >)$ should be $F(A_1) = 1$ - a contradiction.

   So no matter what we end up with a contradiction. Thus our assumption that F is computable must be false.