Homework 5. Due November 29, 9:59PM.

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. Exercise 9.9. Please replace NAND-RAM program with a Turing machine for the problem. That is, consider the case where $F : \{0,1\}^* \to \{0,1\}$ takes two Turing machines P, M as input and $F(P,M) = 1$ if and only if there is some input x such that P halts on x but M does not halt on x. Prove that F is uncomputable. [1 point]

   There are a few different ways to go about solving this problem. Here are two of them.

   Solution 1. This solution makes use of Rice's theorem. Define $F' \colon \{0,1\}^* \to \{0,1\}$ as follows. Given a TM, P, define $F'(P) = F(P, \mathrm{INF})$ where INF denotes the TM which goes into an infinite loop, no matter what input it's given. Note that by definition of F and F', we have $F'(P) = 1$ iff $\exists x \in \{0,1\}^*$ such that $P(x)$ halts.

   Claim: F' is uncomputable.

   Proof. Suppose P and P' are two TMs satisfying $P(x) = P'(x)$ for all $x \in \{0,1\}^*$. Then, clearly $\exists x \in \{0,1\}^*$ such that $P(x)$ halts iff $\exists x \in \{0,1\}^*$ such that $P'(x)$ halts and so $F'(P) = F'(P')$. Thus, F' is semantic and so F' is uncomputable by Rice's theorem. $\square$

Now, recall that by definition we have $F'(P) = F(P, \text{INF})$. Thus, if we had a TM that computed F, then we could use this TM to compute F'. Therefore, since F' is uncomputable, F must be uncomputable as well.

Solution 2. This solution uses a reduction from HALTONZERO. Let N_0 be the following program:

def $N_0(x)$:

(a) If $(x == 0)$: while() {}

(b) Return 1.

That is N_0 is a program that does not halt on 0 but halts on all other inputs. Let M be an input to HALTONZERO and define $\mathcal{R}(M) = (M, N_0)$. We claim that $HALTONZERO(M) = F(\mathcal{R}(M))$.

Case 1: If $HALTONZERO(M) = 1$, then M halts on 0 and N_0 does not halt on 0 so $F(\mathcal{R}(M)) = 1$.

Case 2: If $HALTONZERO(M) = 0$, then for any x either M does not halt or N_0 halts. So $F(\mathcal{R}(M)) = 0$.

The above gives a reduction from $HALTONZERO$ to F. As HALTONZERO is uncomputable, F is uncomputable too.

---

2. Consider the function $EMPTY : \{0,1\}^* \to \{0,1\}$ that takes a DFA as input and outputs 1 if the language of the DFA is empty. That is, $EMPTY(D) = 1$ if D describes a DFA (under some encoidng - the representation is not important) that does not accept any string. Define $EQUIVALENT : \{0,1\}^* \to \{0,1\}$ as the function that takes two DFAs D, D' and checks their equivalence: that is $EQUIVALENT(D, D') = 1$ if $D(x) = D'(x)$, $\forall x$. Give a reduction from EQUIVALENT to EMPTY. [1 point]

   [You can use high-level programming languages or pseudocode to describe your reduction.]

   Solution: Given DFAs D, D', consider the function $f(x) = (D(x) \wedge NOT(D'(x))) \vee (NOT(D(x)) \wedge D'(x))$. Note that, by the closure properties of regular languages, f is regular: (1) NOT(D') is regular (as regular langauges are closed under complement), (2) $D \wedge NOT(D')$ is regular (as regular languages are closed under AND), (3) By same logic, $NOT(D) \wedge D'$ is regular, (4) $(D(x) \wedge NOT(D'(x))) \vee (NOT(D(x)) \wedge D'(x))$ is regular as regular functions are closed under taking OR.

   Therefore, there exists a DFA D'' that computes f as above, and in fact we can compute the DFA for D'' from D, D' (using the arguments we saw for closure operations of regular languages). Define $\mathcal{R}(D, D') = D''$.

   Finally, note that there exists an x such that $D''(x) = 1$ if and only if there exists an x such that $D(x) \neq D'(x)$. Therefore, $EQUIVALENT(D, D') = EMPTY(D'')$.

3. Design a context-free grammar for the following launguage: $L = \{0^m 1 0^n 1 0^{|m-n|} : m, n \geq 0\}$. Here $|m-n|$ denotes the absolute value of $m-n$.

Solution: We can simplify the problem a bit for ourselves by observing that each string in L falls into at least one the following cases: (a) the first block of 0's is at least as large as the second block; (b) the second block of 0's is at least as large as the first block. To formalize this, let

$$L_1 = \{0^m 1 0^n 1 0^{m-n} \colon m \geq n \geq 0\} \text{ and } L_2 = \{0^m 1 0^n 1 0^{n-m} \colon n \geq m \geq 0\}.$$

Clearly $L = L_1 \cup L_2$. We will design grammars separately for L_1 and L_2 and then combine them into a single grammar for L.

Designing the grammar for L_1: Observe that any string in L_1 can be written as $0^k 0^n 1 0^n 1 0^k$ where $n, k \geq 0$ (we are replacing m in the definition of L_1 with $n+k$). Thus we can define the grammar for L_1 as:

$$S \to 0S0|A1,\ A \to 0A0|1.$$

The first rule for S allows us to generate the k 0's on the far left and right of the string, and the second rule for S generates the right-most 1 in the string and transitions us to generating the inner part of the string using A. The first rule for A generates the $2n$ 0's in the string and the second rule for A generates the 1 that separates those 0's into two equally sized blocks.

Designing the grammar for L_2: Observe that any string in L_2 can be written as $0^m 1 0^m 0^k 1 0^k$ where $m, k \geq 0$ (we are replacing n in the definition of L_2 with $m+k$). Thus, we can define the grammar for L_2 as

$$S \to AA,\ A \to 0A0|1.$$

This grammar comes from the observation that any string in L_2 is just the concatenation of two strings of the form $0^\ell 1 0^\ell$ for some $\ell \geq 0$.

Finally, using our grammars for L_1 and L_2, we have the following grammar for L:

$$S \to S_1|S_2,\ S_1 \to 0S_10|A_11,\ A_1 \to 0A_10|1,\ S_2 \to A_2A_2,\ A_2 \to 0A_20|1.$$

Suppose we are trying to generate a string $x \in L$ using the above grammar. The first rule allows us to choose the appropriate grammar based on whether $x \in L_1$ or $x \in L_2$. From there, the second and third rules handle the case of $x \in L_1$ while the fourth and fifth rules handle the case of $x \in L_2$.

---

4. Design a context-free grammar for the following language:

$$L = \{x \in \{0,1\}^* : x \text{ has an equal number of 1's and 0's}\}.$$

You can assume L has the empty string. Write a few sentences to explain your reasoning. [1 point]

[Hint: Think of cases like a) the first and last symbols of x are different; b) the first and last symbols of x are same. In case (a), you can do reduce back to the same question. In case (b), what should you do?]

Solution: Grammar for L: $A \to AA|0A1|1A0|\varepsilon$.

Proof that the grammar above generates L: It is easy to check that every string generated from A has an equal number of 0's and 1's. Further, any string x that has an equal number of 0's and 1's falls into one of two cases: a) the first and last symbol are the same; b) the first and last symbol are different. In case (a), there must be a proper prefix where the number of 0's and 1's is the same so we can use the rule $A \to AA$ to generate the string. In case (b), we can use one of the other two rules $0A1$ or $1A0$.

---

1 Additional Problems

1. Exercise 9.13. Replace NAND-TM with just plain TM in the entire problem. [2 points]

   [Hint: For part (2), try to come up with a program whose description length is at most n but that takes $\omega(TOWER(n))$ steps to stop. I also highly recommend reading the two references in the problem.]

   Solution:

   Proof of part (1): We can use T_{BB} to compute HALTONZERO as follows: If $T_{BB}(P) = 0$, return 0, else return 1. As HALTONZERO is uncomputable, T_{BB} is uncomputable too.

   Proof of part (2): The actual proof does not use much about the TOWER function. The main idea is as follows. Suppose $f : \mathbb{N} \to \mathbb{N}$ is a computable function. Then, there is a Turing machine M_f takes n in binary on its input tape, and takes at least $f(n)$ steps on input n. This is achievable easily for any computable function: You can for instance first compute $f(n)$ and have another for loop that runs for $f(n)$ steps.

   Now, M_f has a fixed size description, say s_f. (It is a program and its description does not depend on the input length.). For every n, consider a new program P_n defined as follows:

   def P_n(x): RETURN $U_{TM}(M_f, n)$, where n is specified in binary and U_{TM} is a fixed constant size universal TM of size say c_u.

   The description length of P_n is $O(c_u) + O(s_f) + O(\log n) = O(1) + O(\log n)$. Thus, for some sufficiently big constant n_f, we would have $|P_n| < n$ for all $n \geq n_f$. This in turn implies that $NBB(n) \geq f(n)$ for all $n \geq n_f$. (As M_f takes at least $f(n)$ steps on input n, P_n takes at least $f(n)$ steps on input 0.)

   The above argument proves that for every computable function $f : \mathbb{N} \to \mathbb{N}$, there exists some constant n_f such that $f(n) \leq NBB(n)$ for all $n \geq n_f$.

Now, returning to the problem, the main point is that $TOWER(n)$, however large it is, is computable. Moreover, even $n \cdot TOWER(n)$ is computable. For instance, if $f(n) = nTOWER(n)$, we can consider the following simple program: **def** nTOWER(n):

(a) Set $a = 1$.
(b) For $i = 1, 2, \ldots, n$, set $a \leftarrow 2^a$.
(c) Set $b = 0$.
(d) For $i = 1, \ldots, na$: $b = b + 1$.

The number of steps taken by the above program on input n is at least $nTOWER(n)$. Further, for the case of the TOWER function as in the problem, the sequence of programs P_n could be the following: **def** $P_n(x)$:

(a) Set $a = 1$.
(b) For $i = 1, 2, \ldots, n$, set $a \leftarrow 2^a$.
(c) Set $b = 0$.
(d) For $i = 1, \ldots, n \cdot a$: $b = b + 1$.

Note that the input to the program P_n is x (which it ignores). The number of steps it takes on input 0 is at least $nTOWER(n)$. The description of length P_n is $O(1) + O(\log n)$ as we did in the general case.

Therefore from our earlier argument exists some n_f such that for all $n \geq n_f$, $nTOWER(n) \leq NBB(n)$. Thus, $\lim_{n\to\infty} TOWER(n)/NBB(n) = 0$ so that $TOWER = o(NBB)$.

---

2. Show that there is a simple constant size TM (or program in your favorite language) `Fermat` such that the program `Fermat` *does not terminate* if and only if the Fermat's last theorem is true (which we know it is ... but don't assume that for this problem - just show equivalence).

Solution: Fermat's last theorem states that no three positive integers a, b, and c satisfy the equation $a^n + b^n = c^n$ for any integer value of $n > 2$. Thus, the idea is to write a program which iterates over all a, b, c, n and terminates if it finds a counterexample. To define the program we'll first define some notation. Given any natural number $k \geq 3$, let

$$T_k = \Big\{(a, b, c, n) \colon a, b, c \in \{1, \ldots, k\}, n \in \{3, \ldots, k\}\Big\}.$$

Note that T_k is a finite set for all k.

def `Fermat`:

(a) $k = 3$
(b) For all $(a, b, c, n) \in T_k$: if $a^n + b^n = c^n$, then terminate.
(c) $k \leftarrow k + 1$. Repeat from step (b).

Clearly, if Fermat's last theorem is true, then this program will never terminate. Now, if Fermat's last theorem is false, then there exists some $a, b, c \geq 1$ and $n \geq 3$ such that $a^n + b^n = c^n$. Let $\ell = \max(a, b, c, n)$ and observe that $(a, b, c, n) \in T_\ell$. Moreover $|T_k|$ is some finite number for all k and so `Fermat` will eventually check the tuple (a, b, c, n) in line (b), and will terminate.

---

3. Exercise 10.1 (4), (6).

   Solution to (4): If F is context free, then H is context free. We can prove this by designing a context free grammar for H as follows. Let (V, R, s) be a context free grammar for F. Let $R' = \{(v, z^R) \colon (v, z) \in R\}$ where z^R denotes the reverse of z. Clearly, a string x can be generated by (V, R, s) if and only if x^R can be generated by (V, R', s), and so (V, R', s) is a context free grammar for H.

   Solution to (6) H is not always context free. Here's a simple counterexample. Let F, G be such that $F(x) = G(x) = 1$ for all $x \in \{0,1\}^*$. Clearly F and G are context free. However, we have $H(x) = 1$ iff $x = uu$ for some $u \in \{0,1\}^*$ and we know this is not context free (see section 10.2.3 of the textbook and the lecture notes).

---

4. Design a context-free grammar for the following languages:

   (a) $L = \{x : \text{ 5'th bit from end is a } 1\}$.

   Solution: $S \to A1BBBB$, $A \to 0A|1A|\varepsilon$, $B \to 0|1$.

   (b) $L = \{x : \ x \text{ has at least three 1's}\}$.

   Solution: $S \to A1A1A1A$, $A \to 0A|1A|\varepsilon$.

   (c) $L = \{0^m 1^n : m \neq n\}$.

   Solution: $S \to 0S_0|S_1 1$, $S_0 \to 0S_0|A$, $S_1 \to S_1 1|A$, $A \to 0A1|\varepsilon$.

   (d) $L = \{x : \ x \text{ is not of the form } 0^n 1^n\}$.

   Solution: For this problem we will make use of our solution for part (c). Observe that if $x \in \{0,1\}^*$ is *not* of the form $0^n 1^n$, then either (i) x is of the form $0^m 1^n$ where $m \neq n$, or (ii) there exists $i < j$ such that $x_i = 1$ and $x_j = 0$ (i.e. x contains a 1 that comes before some 0). We can use the CFG from part (c) to handle case (i). Thus, we have the following CFG for L

$S' \to S|T,\ T \to B1B0B,\ B \to 0B|1B|\varepsilon, S \to 0S_0|S_11,\ S_0 \to 0S_0|A,\ S_1 \to S_11|A,\ A \to 0A1|\varepsilon$

where our start variable is S'. Note that all rules after the rule for B are just copy-paste from part (c) and are there to handle case (i). The second and third rules handle case (ii).
