CSE P531: Computability and Complexity Theory (Spring, 2016)

Homework 3 Out: Thursday, 14-Apr. **Due**: Friday, 22-Apr (9pm in the **Dropbox**)

Reading:

Sipser, Sections 5.1—5.3 and 7.1—7.2

Instructions:

Your proofs and explanations should be clear, well-organized and as concise as possible.

You are allowed to discuss the problems with fellow students taking the class. However, you must write up your solutions completely on your own. Moreover, if you do discuss the problems with someone else, I am asking, on your honor, that you do not take any written material away from the discussion. In addition, for each problem on the homework, I ask that you acknowledge the people you discussed that problem with, if any.

Most of the problems require only one or two key ideas for their solution – spelling out these ideas should give you most of the credit for the problem even if you err in some finer details. So, make sure you clearly write down the main idea(s) behind your solution.

A final piece of advice: Begin work on the problem set early and don't wait until the deadline is only a few days away.

1. Decidability of arithmetical theorems

We will consider the state of true theorems in **modular** arithmetic. Fix a natural number $q > 1$ denote $\mathbb{Z}_q = \{0, 1, \ldots, q-1\}$ and let $\mathrm{Th}(\mathbb{Z}_q, +, \times)$ be the set of true sentences using quantifiers, logical operators, $=$ (equality), and the operations $+$ and $\times$, where the two operations correspond to addition and multiplication modulo q.

For every $q > 1$, describe how one can think about the set of true sentences as a language over some alphabet, and then argue that your language $\mathrm{Th}(\mathbb{Z}_q, +, \times)$ is decidable. In other words, there exists a Turing machine that, when given a sentence as input, accepts the sentence if it is true and rejects the sentence if it false.

The quantifiers are ∀ (for all) and ∃ (there exists), and the logical operators are ∧ (and), ∨ (not), ¬ (negation), and → (implication). For instance, here is the statement that every number has an additive inverse modulo q:

$$\forall x \exists y\ (x + y = 0)$$

This sentence is true. Here is the statement that every number has a multiplicative inverse modulo q:

$$\forall x \exists y\ (xy = 1)$$

This statement is never true (because 0 does not have an inverse). On the other hand, the following statement is a true theorem if and only if q is prime:

$$\forall x\ (\neg(x = 0) \rightarrow \exists y\ xy = 1)$$

Here, q is fixed (it is not part of the input), and your Turing machine should decide whether or not the formula is true.

[It is a fascinating fact that if we work instead over the integers, the corresponding theory $\mathrm{Th}(\mathbb{Z}, +, \times)$ is **undecidable**. This is because there exists a computable reduction that maps an input $\langle M, w\rangle$ to a sentence $\phi_{M,w}$ in $\mathrm{Th}(\mathbb{Z}, +, \times)$ such that M accepts $w \leftrightarrow \phi_{M,w}$ is true. Basically, arithmetic contains a universal Turing machine!]

2. Dynamic programming

Recall that if L is a language, then L^* is the language defined by

$$L^* = \{ w_1 w_2 \cdots w_k : k \geq 0, w_i \in L \text{ for each } i \}$$

In other words, L^* contains strings that are concatenations of zero or more strings in L. The goal of this problem is to prove that if $L \in \boldsymbol{P}$ then $L^* \in \boldsymbol{P}$, where $\boldsymbol{P} = \bigcup_{k \geq 1} TIME(n^k)$ is the set of languages decidable in polynomial time.

This requires an idea known as "dynamic programming." Suppose we are given a string $y_1 y_2 \cdots y_n$ where each $y_i \in \Sigma$. We want to know if the string is in L^*. To do this, we build an $n \times n$ table A, where the entry $A[i,j]$ (for $i \leq j$) is supposed to represent whether the substring $y_i y_{i+1} \cdots y_j$ is in L^*. If we can build this table, then we can just look at the entry $A[1,n]$ to figure out if $y_1 y_2 \cdots y_n \in L^*$.

What's left is to see that we can fill in the table A in polynomial time. Remember that we have assumed $L \in \boldsymbol{P}$, so we have an algorithm that tests membership in L. We can easily fill in some entries of the table: For each $i = 1,2, \dots, n$, we have $A[i,i] = 1$ if the string y_i is in L and $A[i,i] = 0$ otherwise. Now consider the following pseudocode to fill in the rest of A:

```
For $i = 1, 2, \dots, n-1$, do:
    For $j = 1, 2, \dots, n-i$
        $A[j, j+i] = \cdots$
```

Your goal is to fill in $A[i,j]$ using entries of the table A that have already been filled in. The whole algorithm should run in polynomial time, and at the end, we should have $A[i,j] = 1$ if $y_i y_{i+1} \dots y_j \in L^*$ and $A[i,j] = 0$ otherwise. As we said before, $A[1,n]$ then contains the answer to whether $y_1 y_2 \dots y_n \in L$.

Complete the code and argue that it decides membership in L^* in polynomial time.

Extra credit problems [These problems are difficult: They may require some creativity and clever ideas. They are each worth two regular problems. If you get stuck on one, you can ask me for help.]

1. An **oracle** for a language L is an external device that can report whether any string w is a member of L. An **oracle Turing machine** is a modified Turing machine that has the additional capability of querying such an oracle. We write M^L to describe an oracle Turing machine that has an oracle for the language L.

   We say that a language A is **Turing-reducible** to a language B, written $A \leq_T B$, if there is a Turing machine M such that M^B decides A. (In other words, A can be decided with an oracle for B.) As a warm up, you might want to confirm the following two facts:

   1) If $A \leq_T B$ and B is decidable, then A is decidable.
   2) $A_{TM} \leq_T HALT_{TM}$ (in other words, the acceptance language for Turing machines can be decided if we have an oracle for the halting problem)

   Now here's the problem: Show that there are two languages A and B such that $A \nleq_T B$ and $B \nleq_T A$. In other words, A cannot be solved with an oracle for B and B cannot be solved with an oracle for A.

2. Suppose we have n variables $x_1, x_2, \dots, x_n$ and each variable can take only value 0 or 1. We also have an expression of the form

   $$C_1 \cdot C_2 \cdot C_3 \cdots C_m$$

   Where each C_i is of the form $C_i = \max(a_i, b_i)$ and each a_i or b_i is a variable x or $1 - x$. For instance, consider the following expression over the variables x_1, x_2, x_3:

   $$E = \max(x_1, 1 - x_2) \cdot \max(1 - x_1, x_3) \cdot \max(x_1, x_3) \cdot \max(x_2, 1 - x_3)$$

   You are given such a formula as input and the goal is to decide if there exists a setting of the variables to 0/1 values such that the expression equals 1. For example, for E there is a solution:

   $$x_1 = 1, x_2 = 1, x_3 = 1$$

   Plugging these in we get $E = \max(1,0) \cdot \max(0,1) \cdot \max(1,0) = 1 \cdot 1 \cdot 1 = 1$.

   On the other hand, the expression

   $$\max(x_1, 1 - x_2) \cdot \max(1 - x_1, 1 - x_2) \cdot \max(x_1, x_2) \cdot \max(1 - x_1, x_2) = 1$$

   has no solution (you can try all four possible values for x_1, x_2).

   Consider the language of formulas of this form that have a solution. Show that this language is in $\boldsymbol{P}$ (polynomial time).