CS6014-2018 : Computability and Complexity — Max Marks : 42

Problem Set #2 — Due on : Sep 24, 22:00 (hard)

- All submissions must be in PDF form produced using LaTeX. Use the same LaTeXfile (from which this pdf file was produced) to add your information and answers.

Name : Rachit Garg — Roll No: CS14B050

1. (**More on Diagonal Set**)

   Recall
   $$K = \{x \mid M_x \text{ halts on } x\}$$
   Prove the following.

   (a) (5 points) We showed in problem set 1 that K is undecidable. Now show that K is SD-complete.

   (b) (2 points) Show that the proof technique that you have used for part (a) relativises. That is, let $K^X = \{x \mid M_x^X$ is an oracle TM with oracle X and M_x^X halts on $x\}$. Show that K^K is Σ_2-complete.

Solution:

(a) *Proof.* K is undecidable, i.e. $K \notin D$. To show that K is SD-complete, we need to show two properties:

- $K \in SD$

  To prove that K is a semi-decidable language, we need to construct a turing machine M', that accepts x whenever M_x(denoting the turing machine encoded by string s) accepts x. Consider M' to be a universal turing machine that on input x simulates M_x on x.
  Clearly, due to the construction of a universal turing machine, we have that:
  $$M' \text{ accepts } x \iff U \text{ accepts } (M_x, x)$$
  $$M' \text{ accepts } x \iff M_x \text{ accepts } x$$
  Hence K is in SD.

- K is SD-hard

  To show that K is SD-hard, it is sufficient to show that HP many one reduces to K. Note that since HP is SD-hard, this implies K is SD-hard.
  Claim: HP $\leq_m$ K

Consider a function $\sigma : \Sigma^* \to \Sigma^*$
Input: (M, x)
Output: (x')
Consider a machine M', where x' is encoding of M'. We define M' as follows:

```
M′ on input z
______________
Run M on input x
If M ↓, i.e. M halts, then accept z
else reject
```

Clearly, if M halts, then M' accepts all strings, i.e M' accepts x'. If M does not halt, then M' does not accept any string. Hence,

$$M \text{ halts on } x \iff M' \text{ accepts } x'$$
$$M \text{ halts on } x \iff M_{x'} \text{ accepts } x'$$
$$M \text{ halts on } x \iff x' \in \text{K}$$

□

(b) *Proof.* The proof technique relativises easily, we can clearly observe that K^{K} is in Σ_2 as it is semi decidable given an oracle turing machine that solves K. The construction of part(a) remains same and we can construct a universal turing machine having oracle access to K as before.
The second part also follows from part (a) as the reduction holds true here as well, we know from class that HP^{HP} is a Σ_2 complete language, here if we perform the same reduction where M now has access to oracle of HP, we can show that $\text{HP}^{\text{HP}} \leq_m \text{K}^{\text{K}}$. □

2. (**Reductions**) In this problem we study a new notion of reduction between computational problems called *non-adaptive* Turing reduction. A is non-adaptively Turing reducible to B ($A \leq_T^{na} B$), if there is an oracle Turing machine which is total when given oracle access to B and in additional each of the queries (to the oracle) are produced independent of answers to other queries.

   (a) (4 points) Comment and prove the connection between the statements $A \leq_m B$ and $A \leq_T^{na} B$.

   (b) (3 points) Is it true that $A \leq_T^{na} B$ and $B \leq_T^{na} C$, then $A \leq_T^{na} C$? Prove or Disprove.

Solution:

(a) *Proof.* We claim that the relation between the notion of *non-adaptive* Turing reductions is as follows:

$$A \leq_m B \implies A \leq_T^{na} B \implies A \leq_T B$$

Many one reduction is the strongest reduction among the three, it has the extra capability of using only one solver instance and mapping yes and no instances of one problem, directly to a yes and no instance of aother problem. To prove the first implication, we need to provve that there is an oracle Turing machine which is total when given access to B where queries to B are independent of answers to other queries.
Since $A \leq_m B$, there is a reduction σ where σ is total. We can construct an oracle turing machine as follows, on input x and oracle tape to B, we simulate σ on input x, the output received is then checked on the oracle tape of B, if the oracle tape on location $\sigma(x)$ accepts, we accept, else we reject.
Clearly in this oracle turing machine, we only query for one location, thus no other queries are produced and are hence trivially indpendent of answer to other queries.
We also observe that the stronger implication of turing reduction also holds as if a reduction is true in a non adaptive scenario, clearly it is also true for a scenario where queries are allowed to adapt. □

(b) *Proof.* Yes, the relation is transitive.
$A \leq_T^{na} B \implies$ there exists a total OTM M that given oracle access to B, can solve A where the queries to B are independent of each other.
Similarly, $B \leq_T^{na} C \implies$ there exists a total OTM N that given oracle access to C, can solve B where the queries to C are independent of each other.
Given the two total OTM's M and N, we can construct a total OTM M' where M' is given oracle access to C, an input x. The turing machine functions as follows. It simulates M on input x where whenever an oracle access to B is made at position y, the turing machine simulates N on y using the oracle tape C. It outputs whatever the result of simulation of M is. Clearly, all the operations here are total as the turing machines M and N are total and the machine M' can solve A as M solves A.
The reduction is also non adaptive. Let on input x the list of queries made to B during simulation of B be $\{Q_1, \ldots, Q_m\}$. Each query Q_i is realized using queries to oracle of C during simulation of N, let they be given by a lits of queries $\{Q_{i,1}, \ldots, Q_{i,n}\}$. During our simulation of M', we make the following queries, $Q = \{Q_{1,1}, Q_{1,2}, \ldots, Q_{m,n}\}$. Since queries done in simulation of N are independent, and the queries done in simulation of M are independent, we have that all the queries in Q are independent. □

3. ($T(\mathbb{N})$ **is beyond AH**) Recall the set of true sentences which we defined when we

discussed the computability based proof of the Gödel's incompleteness theorem.

$$T(\mathbb{N}) = \{\phi \mid \phi \text{ is a sentence which is true }\}$$

(a) (4 points) A quantified expression (called a sentence) is said to be a Σ_k-sentence it is a predicate with k quantifiers and no free variables. Let us define,

$$T_k(\mathbb{N}) = \{\phi \mid \phi \text{ is a } \Sigma_k \text{ sentence which is true }\}$$

Show that T_k is complete for Σ_k.

(b) (3 points) Use part (a) to argue that $T(\mathbb{N})$ cannot be in the arithmetic hierarchy. (Hurray ! - beyond AH !!). (*Hint : Use the fact that* Σ_k *is strictly contained in* Σ_{k+1}).

Solution:

(a) *Proof.* To show that T_k is complete for Σ_k, we need to show membership in Σ_k and show that T_k is harder than all languages in Σ_k.

- $T_k \in \Sigma_k$

  We prove this using induction. Base case is trivially true. For the induction step, we assume that $T_{k-1} \in \Sigma_{k-1}$, we show that there is an OTM (not necessarily total), that given oracle access to a language in Σ_{k-1} can be used to solve T_k.

  Given as input a Σ_k sentence given by $\phi = (\exists y_1)(\forall y_2) \ldots P(y_1, y_2, \ldots, y_k)$, we have access to an oracle for a language in Σ_{k-1}, let that language be T_{k-1}. We can now create an OTM M, where M searches among all possible y_1, and tries to answer queries regarding $(\forall y_2) \ldots P(y_1, y_2, \ldots, y_k)$ using T_{k-1}. Note that we can solve queries such as $(\exists y_2) \ldots \neg P(y_1, y_2, \ldots, y_k)$ using direct oracle access to T_{k-1}. We can then use the answer of this query to answer our query and find out if there exists a y_1 that can solve ϕ.

- T_k is Σ_k-hard

  To Show: $\forall A \in \Sigma_k, A \leq_m T_k$

  Consider a function $\sigma : \Sigma^* \to \Sigma^*$

  Input: (x)

  Output: (x')

  From class we know that since, $A \in \Sigma_k$, there exists, a predicate P such that $A = \{x | (\exists y_1)(\forall y_2) \ldots P(x, y_1, y_2, \ldots, y_k)\}$. The reduction outputs x' where $x' = (\exists y_1)(\forall y_2) \ldots P(x, y_1, y_2, \ldots, y_k)$.

  If $x \in A$ implies that there exists proper quantifiers such that the Σ_k sentence x' is true.

If $x \notin A$, implies that the Σ_k sentence x' is not true. Hence the reduction is valid.

$$x \in A \iff x' \in T_k$$

□

4. (7 points) (**Should production function be total?**) Recall the production function σ that we defined in class for a productive set. We insisted that the function should be total. In this exercise we will remove that restriction. A set P is *partially productive* if there is a partial recursive function(i.e. computed by a Turing machine N - which need not be total) σ such that:

$$\forall x \quad (\mathcal{W}_x \subseteq P \implies N \text{ halts on } x \ \& \ \ \sigma(x) \in P \setminus \mathcal{W}_x)$$

Show that any partial productive set P also has a recursive production function.

Extra credit (5 Marks)[1] Argue that the production function can even be made a one-one recursive function.

Solution:

Proof. The main observation to solve this question is that we have a turing machine N(not necessarily total) that halts on x, if $\mathcal{W}_x \subseteq P$. And N might not halt on x, if $\mathcal{W}_x \not\subseteq P$. The condition $\mathcal{W}_x \not\subseteq P$ tells us that there is some string y where $y \in \mathcal{W}_x$ and $y \notin P$, i.e. $\overline{P} \cap \mathcal{W}_x \neq \phi$.
If $P = \overline{\mathrm{K}}$.
We construct a total turing machine M'. On input x we run in parallel steps (a) N on x, (b) on all inputs y, W_x on y and K on y in a diagoanl manner. M' accepts if N halts on x or there is some y where W_x and K both halt on y, this happens in our scenraio because of gurataanteed existence of y if N does not halt. When the latter happens, we ouput the same string x. Hence M' is total and satisfies the production function criteria.
This construction can be generalized to whenever $\overline{P}$ is semi-decidable. Note that this is not a necessary requirement and this argument fails if there is no turing machine for $\overline{P}$. □

5. (7 points) (**Effectively Simple Sets**) A simple set is *effectively simple* if there is a recursive[2] function f such that:

$$\forall n \in \mathbb{N} : \mathcal{W}_n \subseteq \overline{A} \implies |\mathcal{W}_n| \leq f(n)$$

[1] As a policy, all extra credit marks will be added only in the end, after the grade cut offs are decided, and hence does not affect the grades of the students who do not attempt these questions, but it can help individuals to jump up to the higher grade. Scaling factor would remain the same.

[2] computable by a total Turing machine

Show that Post's simple set is effectively simple.

Extra credit (5 Marks)[1] show that if a set A is effectively simple, argue that $K \leq_T A$ (that is, K is decidable in A). In class we had shown that simple set cannot be SD-complete (that is $K \not\leq_m A$). This justifies why Friedberg-Muchnik theorem (which we will state in class) cannot be proved by using Post's Simple Set and hence requires a different proof (Read Lecture 38, Page 253, in Kozen2 Book, if you are interested).

Solution:

Proof. We observe that if A is simple implies that it intersects non trivially with every infinite semi-decidable language. Since $\mathcal{W}_n \subseteq \overline{A}$, this implies that $\mathcal{W}_n$ must not be an infinite semi-decidable language, i.e. $\mathcal{W}_n$ must be finite.
For Post's simple set we have that if $\mathcal{W}_n$ is finite and outputs a string w such that $|w| > 2|x|$, then w will be outputted by the Enumerating Turing Machine and hence will be included in A due to the construction of Post's simple set. Hence any finite $\mathcal{W}_n$ must only accept strings with length less than equal to $2|x|$. This implies that $|\mathcal{W}_n| \leq \Sigma^{2n}$. Hence $f(n)$ is a total recursive function. □

6. (7 points) (**Productivity and Many-one Reductions**)
   (a) (3 points) In class, we showed that if $K \leq A$ then $\overline{A}$ is productive. This can also be seen as : since $\overline{K} \leq_m \overline{A}$ and $\overline{K}$ is productive and hence $\overline{A}$ is productive. Extend this proof to argue: *If $A \leq_m B$ and A is productive, then B is also productive.*
   (b) (4 points) If A is productive, B is semi-decidable, and $B \subseteq A$, show that $A - B$ is productive.

Solution:

(a) *Proof.* Let A be productive via a total function σ. Since $A \leq_m B$, there exists a total function τ such that instances of A get converted to instances of B on application of τ. We claim that there is a production function σ' for B.
Claim: $\sigma' = \tau(\sigma(\delta))$ where δ is the same function defined in class, i.e. on input i, $\delta(i)$ is the code for the following machine M'.

```
$M'$ on input $x$
________________
Apply $\tau$ on x to calculate $\tau(x)$.
Run $M_i$ on input $\tau(x)$
Accept if it accepts.
```

Clearly the function σ' is total as the functions τ, σ, δ are all total. We now need to prove correctness.

To Prove: $\forall i, \mathcal{W}_i \subseteq B \implies \sigma'(i) \in B - \mathcal{W}_i$

$$
\begin{aligned}
\forall i, \mathcal{W}_{\delta(i)} \subseteq A &\implies \sigma(\delta(i)) \in A - \mathcal{W}_{\delta(i)} \\
\forall i, \mathcal{W}_{\delta(i)} \subseteq \tau^{-1}(B) &\implies \sigma(\delta(i)) \in \tau^{-1}(B) - \mathcal{W}_{\delta(i)} \\
\forall i, \tau^{-1}(\mathcal{W}_i) \subseteq \tau^{-1}(B) &\implies \sigma(\delta(i)) \in \tau^{-1}(B) - \tau^{-1}(\mathcal{W}_i) \\
\forall i, \tau^{-1}(\mathcal{W}_i) \subseteq \tau^{-1}(B) &\implies \sigma(\delta(i)) \in \tau^{-1}(B - \mathcal{W}_i) \\
\forall i, \tau^{-1}(\mathcal{W}_i) \subseteq \tau^{-1}(B) &\implies \tau(\sigma(\delta(i))) \in B - \mathcal{W}_i \\
\forall i, \mathcal{W}_i \subseteq B &\implies \tau(\sigma(\delta(i))) \in B - \mathcal{W}_i
\end{aligned}
$$

Hence Proved □

(b) *Proof.* Since A is productive, we know there exists a production function σ such that $\forall i, \mathcal{W}_i \subseteq A \implies \sigma(i) \in A - \mathcal{W}_i$.
Let B be semi-decidable via a turing machine given by $\mathcal{W}_{x'}$.
We know from class that union of two semi-decidable languages is a semi-decidable language. Using the same idea we procees, let δ be a function that is defined as follows. On input i, let $\delta(i)$ be the code for the following machine M'.

```
$M'$ on input $x$
________________
Run $M_i$ on input $x$ and $M'_x$ on input $x$ step by step
Accept if either accepts
```

Clearly this is a machine that accepts the union of the language accepted by M'_x and M_x.

$$
\begin{aligned}
L(M') &= L(M'_x) \cup L(M_i) \\
\mathcal{W}_{\delta(i)} &= \mathcal{W}_{x'} \cup \mathcal{W}_i
\end{aligned}
$$

Consider a function $\sigma' : \Sigma^* \to \Sigma^*$ for showing $A - B$ is productive. We claim that the composition function of σ and δ i.e. $\sigma' = \sigma \circ \delta$ is a valid production function for $A-B$. Clearly the composition is total as σ and δ are total. Proving correctness.

$$
\begin{aligned}
\forall i, \mathcal{W}_i \subseteq A - B &\implies \mathcal{W}_i \subseteq A - \mathcal{W}_{x'} \\
\forall i, \mathcal{W}_i \subseteq A - B &\implies \mathcal{W}_i \cup \mathcal{W}_{x'} \subseteq A \\
\forall i, \mathcal{W}_i \subseteq A - B &\implies \mathcal{W}_{\delta(i)} \subseteq A \\
\forall i, \mathcal{W}_i \subseteq A - B &\implies \sigma(\delta(i)) \in A - \mathcal{W}_{\delta(i)} \\
\forall i, \mathcal{W}_i \subseteq A - B &\implies \sigma(\delta(i)) \in A - (B \cup \mathcal{W}_i) \\
\forall i, \mathcal{W}_i \subseteq A - B &\implies \sigma(\delta(i)) \in (A - B) - \mathcal{W}_i
\end{aligned}
$$

Hence proved. □