Computability and Complexity theory
Assignment # 2

May 5, 2023

Roy, Akash | CS22M007
M.Tech CS | Indian Institute of Technology, Madras

1. Recall the definition of $\overline{HP}$ defined in the class. Let $\mathsf{HP}' = \Sigma^* \setminus HP$ (i.e. the set-theoretic compliment). Show that $\overline{HP}$ is recursively enumerable if and only if HP' is recursively enumerable.

Ans: $\overline{\mathbf{HP}}$ defined in the class was $\{M\#x \mid$ M does not halt on x$\}$. We can write the set-theoretic compliment HP as $\overline{\mathbf{HP}}$ defined in the class was $\{M\#x \mid$ M does not halt on x$\} \bigcup A$ where $A = \{M\#x \mid$ M is not a valid encoding of turing machine and x is some arbitrary string$\}$.

Now from the question we need to show that

$$\overline{\mathbf{HP}} \in \mathbf{REL} \iff \Sigma^* \setminus HP \in \mathbf{REL}$$

Forward direction: We get that $\overline{\mathbf{HP}} \in \mathbf{REL}$ so there exists a non total turing machine $\mathbb{M}$ accepting $\overline{\mathbf{HP}}$. So for all the strings in $A = \{M\#x \mid$ M is not a valid encoding of turing machine and x is some arbitrary string$\}$ we define a fixed turing machine $\hat{M}$ such that no matter what the input is $\hat{M}$ is always accepting. We begin by constructing a new turing machine M^{α} that simulates $\mathbb{M}$ on x for input $M\#x$ where M is valid and if some $M\#x \in A$ comes where M is not a valid encoding of a turing machine and x some random input we convert that to $\hat{M}$ and accept x. So then $\overline{\mathbf{HP}} \bigcup \mathbf{A}$ also becomes **REL**. $\implies \Sigma^* \setminus HP \in \mathbf{REL}$

Backward direction It is in the statement that $\Sigma^* \setminus HP \in \mathbf{REL}$ and our class defined $\overline{\mathbf{HP}}$ is a subset of $\Sigma^* \setminus HP \in \mathbf{REL}$. Any subset of **REL** will always will be **REL**.

Hence proved $\overline{\mathbf{HP}} \in \mathbf{REL} \iff \Sigma^* \setminus HP \in \mathbf{REL}$.

2. Show that the relation $\leq_m$ is transitive, i.e., if $A \leq_m B$, and $B \leq_m C$, then $A \leq_m C$.

Ans: It is given that $A \leq_m B$ holds good, this implies there exists some function $f : \Sigma^* \to \Delta^*$ for which

$$\forall x \in A \iff f(x) \in B$$

and f is computable.

So a similar statement is also true for $B \leq_m C$. So there exists some function $g\colon \Sigma^* \to \Delta^*$ for which

$$\forall x_1 \in B \iff g(x_1) \in C$$

and g is computable.

So now let's take an element k from set **A**. For k present in A there must be some element j present in **B**, and for that element $j \in B$ there must exists an element $l \in C$. So for every element $k \in A$, there exists an element $l \in C$.

Similar statement can be said for non-membership also.

$$\forall x \notin A \iff f(x) \notin B \,\&\, \forall x_1 \notin B \iff g(x_1) \notin C$$

The two statements up above means membership in A is many to one reducible to membership in C.

So we can draw the conclusion that if $A \leq_m B$ and $B \leq_m C$ then $A \leq_m C$.

3. Show that if a language $A \subseteq \{0,1\}^*$ is recursive, then so is the concatenation $AA \stackrel{\triangle}{=} \{xy \mid x, y \in A\}$.

Ans: This problem is equivalent to proving whether recursive languages are closed under operation concatenation?

Given that $A \subseteq \{0,1\}^*$ is recursive means there exists a total Turing machine M accepting A. We construct a new Turing machine $\hat{M}$ in the following way

- For each input y we do the followings, say size of input is n,
- **RUN 1:** We run $y[0]$ & $y[1 \to n]$ on Turing machine M. M either accepts or rejects $y[0]$ and $y[1 \to n]$. We define the behaviour of turing machine $\hat{M}$ the following way that if both $y[0]$ and $y[1 \to n]$ is rejected by total turing machine M, we go to **RUN: 2**. If either one of them is accepted and other is rejected that means the concatenation is not good we concatenated $x \in A$ with something $y \notin A$, so we reject. Accept if both is accepted means we've found an xy such that $x \in A \,\&\, y \in A$, where $x = \text{input}[0] \,\&\, y = \text{input}[1 \to n-1]$

- **RUN 2:** We run $y[0 \to 1]$ & $y[2 \to n]$ on Turing machine M. M either accepts or rejects $y[0 \to 1]$ & $y[2 \to n]$. If both are rejected we go to **RUN 3**. Same otherwise, reject if one is accepted and other is rejected (**NOT** a valid concatenation). Accept if both is accepted.
- So on...
- At **RUN K** we check if $y[0 \to K]$ & $y[K+1 \to n]$ is accepted. Same as before if either is accepted and other is rejected then reject, if both of them is accepted accept and if both are rejected go to **RUN K + 1**,
- We stop when we check $y[0 \to n-1]$ & $y[n]$. If these 2 are not accepted, if one is accepted and one is rejected we Reject and halt, if both are accepted we accept.

Language of this turing machine $\hat{M}$ is $\{xy \mid x \in A \,\&\, y \in A\} = AA$. If we see for all $y \in \Sigma^*$ $\hat{M}$ is always halting. So $\hat{M}$ is total turing machine. So concatenation language AA is recursive.

4. Prove (without using Rice's theorem) that the set

$$\{(M_1, M_2) \mid \text{ there is some input where both } M_1 \text{ and } M_2 \text{ halt}\}$$

is not recursive.

Ans: Say $D =$

$$\{(M_1, M_2) \mid \text{ there is some input where both } M_1 \text{ and } M_2 \text{ halt}\}$$

We begin the proof by reduction from H_P. So there should be some computable function f such that $H_p \leq_m D$

We construct $\hat{M}$. $\hat{M}$ takes input $y \in \Sigma^*$. $\hat{M}$ halts on y if and only if M halts on x.

So construction of $\hat{M}$ and Let M be some turing machine,

- Take input $y \in \Sigma^*$
- Erase y
- Write x on input tape

- Run M on x
- If M halts on x, halt $\hat{M}$

M halts on x implies $M\#x \in H_P$. Now for all input $y \in \Sigma^*$ $\hat{M}$ halts y if M halts on x. When M does not halt on x we don't accept anything so $L(\hat{M}) = \phi$

So $M\#x \in H_P \iff f(M\#x) \in D$

$\implies$ y is some input where $\hat{M}, \hat{M}$ halts. $\implies$ $(\hat{M}, \hat{M}) \in D$.

So $H_p \leq_m D$. So D is not recursive, it is recursively enumerable.

5. An unreachable state is a state which can never be reached from the start state on any input. Show that the language

$$\{M\#q \mid q \text{ is an unreachable state of } M \text{ on any input}\}$$

is not recursive. (Your proof should NOT use Rice's theorem.)

Ans: We begin by constructing a new Turing machine $\hat{M}$. We know what are all its accepting states, rejecting states, say this is a non-total turing machine.

We define a new language **REACHABLE** = {M#q | q is reachable state of M on any input}. Then our question becomes the language $\overline{\textbf{REACHABLE}}$.

We assume that **REACHABLE** is decidable hence recursive. Let's construct one more Turing Machine M_B, it'll $\forall x \in \Sigma*$ will run $\hat{M}$ on x. We know what are the accepting states in turing machine $\hat{M}$. So running $\hat{M}$ on x if we see that for input x accepting states are not reachable (because we assumed REACHABLE is decidable) we halt. So $\forall x \in \Sigma*$ our turing machine $\hat{M}$ becomes a total turing machine which contradicts with our construction. Hence our assumption **REACHABLE** is decidable hence recursive is not true.

REACHABLE is not recursive so it may be $\in$ REL or $\in$ Non-REL. (REL means recursively enumerable). So complement of **REACHABLE** which is our question will not be $\in$ REC. In particular if **REACHABLE** is in REL then $\overline{\textbf{REACHABLE}}$ must not be REL. But whatever is true $\overline{\textbf{REACHABLE}}$ is never recursive.