Theory of Computation Assignment no. 10

Goktug Saatcioglu

(1) Let A be a language such that $A \in Co-RE$ which means that $A^c \in RE$. This means that the complement of A can be identified/recognized by a Turing machine M'. Furthermore, the complement of A is all words such that $w \in A^c$ then M' accepts and $w \notin A^c$ then M rejects or loops forever. Let M be another Turing machine such that we check if $w \notin A$ (meaning $w \in A^c$) and if $w \in A$ (meaning $w \notin A^c$). Thus, M is an algorithm that simulates the running of a word w on M for $i = 1, 2, 3, \ldots, \infty$ steps and if at any point M accepts then M' rejects and if at any point M rejects then M' accepts. Also, if M loops forever then M' will also loop forever. From this we see that if $w \notin A$, then M' will accept since if $w \notin A$ then M will reject which will mean M' will accept and $w \in A^c$. Similary, if $w \in A$ then M' will reject since if $w \in A$ then M will accept which will mean M' will reject $w \notin A^c$. Finally, if M loops forever then we do not know if $w \in A$ or $w \notin A$. However, since we assume $A \in Co-RE$ we must assume that we can identify all $w \notin A$. This means that M may also loop forever if $w \in A$ which means that M' also loops forever for $w \notin A^c$. Thus, the forward direction has been proved ($\implies$). For the backward direction ($\impliedby$) where we first assume the existence of such a machine M (as described in the question), we can simply create M' by using the algorithm above. Since M' using this construction decides all words $w \notin A$ then we know that we have decided all words $w \in A^c$ (and have not decided $w \in A$ and $w \notin A^c$) which means we identify/recogize A^c. This then implies that $A \in Co-RE$.

(2)

a. If $A \preceq B$ then there exists a truth preserving reduction from A to B. Furthermore, if $B \preceq C$ then there exits a truth preserving reduction from B to C. Thus, we know that there is an algorithm M_1 (i.e. Turing machine) that takes an input word x and outputs a word y such that if $x \in A$ then $y \in B$ and if $x \notin A$ then $y \notin B$. We also know that there is an algorithm M_2 (i.e. Turing Machine) that takes an input word y and outputs a word z such that if $y \in B$ then $z \in C$ and if $y \notin B$ then $z \notin C$. Combining M_1 and M_2 together, we can create an algorithm M_3 (i.e. Turing machine) that takes an input x and first simulates M_1 to get an output y. Then M_3 will simulate M_2 on y to get z. Firstly, since M_1 and M_2 we know that M_3 is decidable. Secondly, we see that if $x \in A$ then $y \in B$ then $z \in C$ (or if $x \notin A$ then $y \notin B$ then $z \notin C$) which implies that $x \in A$ then $z \in C$ (or $x \notin A$ then $z \notin C$) which proves that $\preceq$ is transitive. Thus, we conclude if $A \preceq B$ and $B \preceq C$ then $A \preceq C$.

b. We begin by proving if $A \preceq B$ then $A^c \preceq B^c$. If $A \preceq B$ then there exists a truth preserving reduction from A to B which means that we know that there is an algorithm (i.e. Turing machine) that takes an input word x and outputs a word y such that if $x \in A$ then $y \in B$ and if $x \notin A$ then $y \notin B$. We then note that $x \in A \iff x \notin A^c$ and $y \in B \iff y \notin B^c$. Thus, we have proved that if $x \notin A^c$ then $y \notin B^c$ and if $x \in A^c$ then $y \in B^c$ which completes the proof. We conclude that if $A \preceq B$ then $A^c \preceq B^c$. Next, we note that if $B \in Co-RE$ then $B^c \in RE$. Since if $A \preceq B$ then $A^c \preceq B^c$ (from the above proof) and $B^c \in RE$ then $A^c \in RE$ which means that $A \in Co-RE$. Thus, we conclude if $B \in Co-RE$ and $A \preceq B$ then $A \in Co-RE$.

c. This is simply the contrapositive of the statement from part b. Thus, it follows that from b. that if $A \notin Co-RE$ and $A \preceq B$ then $B \notin Co-RE$ since if $B \in Co-RE$ then $A \in Co-RE$ which would contradict our assumption $A \notin Co-RE$. Thus, we conclude if $A \notin Co-RE$ and $A \preceq B$ then $B \notin Co-RE$.

d. We know that $HALT \notin Co-RE$ since we know $HALT \in RE$ and if $HALT \in Co-RE$ were to be true this would imply that $HALT \in R$ which we know is false (because we know that

$HALT \notin R$) giving us a contradiction. Thus, by using c. we conclude that $HALT \notin Co - RE$ and $HALT \preceq ALWAYS\_HALT$ implies that $ALWAYS\_HALT \notin Co - RE$.

(3) a. We need to show a truth preserving reduction from $HALT$ to $ACCEPT$ where we have an algorithm M_f that takes an input word x and outputs a word y such that if $x \in HALT$ then $y \in ACCEPT$ and if $x \notin HALT$ then $y \notin HALT$. Consider the following algorithm M':

$M'(\langle M \rangle, \langle w \rangle)$ (where M is a Turing machine, w is an input word)

1. Run M on w.
2. If M accepts when run on w, then M' accepts.
3. If M rejects when run on w, then M' accepts.

Then we create the algorithm M_f as follows:

$M_f(\langle M \rangle, \langle w \rangle)$ (where M is a Turing machine, w is an input word)

1. Construct the Turing machine M'.
2. Output $\langle M' \rangle, \langle w \rangle$.

Now if an input $w \in HALT$ then at one point M must halt which means that it either rejects or accepts. When either happens then M' accepts which means $w \in ACCEPT$. Similarly, if an input $w \notin HALT$ then at no point M will halt which means that M' will not halt (since step 1 will imply that M' also runs forever) which means that $w \notin ACEEPT$. Thus, M_f is a truth preserving reduction from $HALT$ to $ACCEPT$ and we conclude that $HALT \preceq ACCEPT$.

b. From a. we know that $HALT \preceq ACCEPT$. Using (2)c. we also know that if if $A \notin Co - RE$ and $A \preceq B$ then $B \notin Co - RE$. Thus, we begin by noting that $HALT \notin Co - RE$ since if $HALT \in Co - RE$ were to be true then $HALT \in R$ would be true because we also know that $HALT \in RE$. This leads to a cotnradiction since $HALT \notin R$ which then means that we can deduce that $HALT \notin Co-RE$. Then using the statement from (2)c., since $HALT \notin Co-RE$ and $HALT \preceq ACCEPT$ we know that $ACCEPT \notin Co - RE$. Thus, we conclude that $ACCEPT \notin Co - RE$.

(4) We know that the language A has an enumerator say E. Then we can construct a Turing machine M that takes in an input word w and recognizes A as follows:

```
M(⟨w⟩):
  for each w′ enumerated by E      (i.e. we run E and every time E outputs a word w′)
    if w′ equals w then accept     (i.e. w appears in the output of E)
    endif
  endforeach
end
```

Thus, if $w \in A$ then it must be enumerated by E meaning at one point w is equal to w' meaning we accept. If $w \notin A$ then it will not be enumerated by E meaninig at no point will w be equal to w' meaning we never accept. If A is finite and $w \notin A$ then the for loop for E is finite so we can either reject after the for loop completes or loop infinitely. If A is infinite then A will run infinitely as long as $w \notin A$. Either way, we see that we accept only if $w \in A$ meaning M recognizes A. Therefore, we conclude that if a language A has an enumerator, then $A \in RE$.

(5) a. If $L \in R$ then there exists a Turing machine M that decides wether a word $w \in \Sigma^*$, where Σ is the alphabet for M, is in L by either accepting or rejecting w. Thus, we can enumerate the words of Σ^* in lexigraphic order using a Turing machine E and then testing each word by running M. If M accepts (meaning $w \in L$) then we print the word and if M rejects (meaning $w \notin L$) then we move onto the next word. From this we see that E prints all the words in L such that shorter words are always printed before longer words. We note that E may possibly run an infinite amount of time but the property will still hold. Therefore, we conclude that if $L \in R$ then it is nicely enumerable.

b. If L is nicely enumerable then there exists a Turing machine E that prints the words in L in lexicographic order. We split the language L to two cases. For case one, if L is finite then there exists a Turing machine M that can decide it by simply "hardcoding" every word in L (which is possible since L is finite) into M and then running M. Thus, if L is nicely enumerable and L is finite then $L \in R$. For the second case, if L is infinite then we consider a Turing machine M that decides L as follows. Upon receiving an input w, M will use E to start enumerating all the words in L until some word w' that occurs lexicographically later than w appears. This is bound to happen since L is infinite. If w appeared in the enumeration before w' occurred then M will accept and if w has not appeared in the enumeration before w' occurred then M will reject. A check for whether the next lexicographic word after w has appeared can be done by checking whether each word enumerated by E is the same as w and if an enumerated word is w then the next word must be w'. This way even if L is infinite we see that we can decide whether a word is in L and we can say that if L is nicely enumerable and L is infinite then $L \in R$. Combining the two cases together, we conclude if L is nicely enumerable then $L \in R$.

(6) a. We begin by noting that if M when run on λ (i.e. $M(\lambda)$) does halt at some point it must do so in some n number of steps after M starts running on λ. Furthermore, since the halting problem is undecidable we know that this problem is also undecidable (if it weren't we could also solve $HALT$ using M). Thus, we must simulate M on λ for some n steps. Consider the following algorithm M_w:

$M_w(\langle M \rangle, \langle w \rangle)$: (where M is a Turing machine, w is an input word)

1. $n = |w|$.
2. Simulate $M(\lambda)$ for n steps.
3. If M halts (accepts or rejects) on λ in n steps then M_w rejects.
4. If M doesn't halt (doesn't accept or reject) on λ in n steps then M_w accepts.

Then we create the algorithm M' as follows:

$M'(\langle M \rangle, \langle \{w\} \rangle)$: (where M is a Turing machine, w is set of input words)

1. Construct the Turing machine M_w.
2. If $\forall x \in w$ $M_w(\langle M \rangle, \langle x \rangle)$ accepts then M' halts
3. If $\exists x \in w$ where $M_w(\langle M \rangle, \langle x \rangle)$ rejects then M' does not halt (by going into an infinite loop)
4. Output $\langle M' \rangle, \langle w \rangle$.

And then we create the reduction algorithm M_f as follows:

$M_f(\langle M \rangle, \langle \{w\} \rangle)$: (where M is a Turing machine, w is set of input words)

1. Construct the Turing machine M' (which implicilty constructs the Turing machine M_w).
2. Output $\langle M' \rangle, \langle w \rangle$.

Thus, we see that $\langle M_w \rangle$ is the description of the Turing machine M_w that satisfies the properties as asked in the question. Furthermore, M_f is the reduction algorithm we will use to for b. to prove that $HALT_\lambda^c \preceq ALWAYS\_HALT$. Note that M_f also uses an intermediary description as given by M'.

b. To show that $HALT_\lambda^c \preceq ALWAYS\_HALT$ we need to show a truth preserving reduction from $HALT_\lambda^c$ to $ALWAYS\_HALT$ where we have an algorithm M_f that takes an input word x and outputs a word y such that if $x \in HALT_\lambda^c$ then $y \in ALWAYS\_HALT$ and if $x \not\in HALT_\lambda^c$ then

$y \notin ALWAYS\_HALT$. This algorithm is already given by M_f as described in a., thus we only need to prove that it has the truth preserving property. If M' never enters an infinite loop (meaning M_w always halts by always rejecting) then this means that M' always halts when run on $\{w\}$. Thus, if $\{w\} \in HALT_\lambda^c$ then $\{w\} \in ALWAYS\_HALT$. Conversely, if M' enters an infintie loop at some point when run on $\{w\}$ then M' will never halt since M_w must have accepted at some point which means that $\{w\} \notin HALT_\lambda^c$ and in turn $\{w\} \notin ALWAYS\_HALT$. Thus, the algorithm M_f is a truth preserving reduction from $HALT_\lambda^c$ to $ALWAYS\_HALT$.

c. We begin by noting that $HALT_\lambda \in RE$ as we can recognize it in a similar way we recognize $HALT$ by using the same Turing machine that identifies $HALT$ but modifying it such that it only runs on the empty input and accepts only if its halts on λ. This then implies that $HALT_\lambda \notin Co-RE$ which by definition means $HALT_\lambda^c \notin RE$ since if $HALT_\lambda \in Co-RE$ were to be true then $HALT_\lambda \in R$ would be true whcih then leads to a contradction since we know that $HALT_\lambda \notin R$. (Note: The fact that $HALT_\lambda \notin R$ was shown in class and is thus not proved here.) Now assume for the sake of contradiction that $ALWAYS\_HALT \in RE$. From the property that states that if $A \preceq B$ and $B \in RE$ then $A \in RE$ this would mean that $HALT_\lambda^c \in RE$ which in turn means that $HALT_\lambda \in Co-RE$. This is a contradiction to our initial claim that $ALWAYS\_HALT \in RE$ because we know that $HALT_\lambda \notin Co-RE$ (and $HALT_\lambda^c \notin RE$). Thus, we conclude that $ALWAYS\_HALT \notin RE$.