CSE200 Lecture Notes – Recursive Enumerable Languages and Computable Languages

Lecture by Russell Impagliazzo
Notes by Jiawei Gao
Adapted from notes by William Matthews

January 21, 2016

1 Classes R.E. and co-R.E.

Previously we have proved that the set of computable problems is the same under different computation models including 1-tape TM, k-tape TM, and RAM. Similarly we have seen the set of polynomial-time solvable problems (denoted by P) is also equal under these different computation models.

We also introduced Time Hierarchy. If we define

$$\mathsf{QP} = \bigcup_k \mathsf{TIME}(2^{\log^k n})$$
$$\mathsf{EXP} = \bigcup_k \mathsf{TIME}(2^{n^k})$$
$$\mathsf{EE} = \bigcup_k \mathsf{TIME}(2^{2^{n^k}})$$
$$\cdots$$

We have

$$\mathsf{P} \subsetneq \mathsf{QP} \subsetneq \mathsf{EXP} \subsetneq \mathsf{EE} \subsetneq \mathsf{EEE} \cdots \subseteq \mathsf{COMP} \subseteq \mathsf{R.E.}$$

where COMP is the set of computable languages, and R.E. is the set of *recursive enumerable* languages.

Definition 1.1. A language L is in the class R.E. if there exists a Turing machine M such that for all $x \in L$, $M(x)$ halts and accepts, and for all $x \notin L$, $M(x)$ does not accept (it may reject, or it may not halt.)

Let $HALT = \{(M, x) \mid M \text{ eventually halts on } x\}$.

Proposition 1.1. $HALT \in$ R.E..

Definition 1.2. A language L is in the class co-R.E. if $\overline{L} = \{x \mid x \notin L\}$ is in R.E..

Definition 1.3. A language L is in the class COMP (or *recursive*) if there exists a Turing machine M such that for all $x \in L$, M halts and accepts, and for all $x \notin L$, $M(x)$ halts and rejects.

Proposition 1.2. $HALT \notin$ COMP.

Theorem 1.3. COMP $=$ R.E. $\cap$ co-R.E..

Proof sketch. Here we prove one direction: if $L \in$ R.E. and $L \in$ co-R.E. then $L \in$ COMP. Let M_1 be a TM so that if $x \in L$, M_1 halts and accepts; if $x \notin L$, M_1 either rejects or never halts. Let M_0 be a TM so that if $x \notin L$, M_0 halts and accepts; if $x \in L$, M_0 either rejects or never halts.

We run the following algorithm:

1. $T \leftarrow 1$
2. While not done, do
   (a) Run M_0 and M_1 for T steps.
   (b) If M_0 accepts, reject and done.
   (c) If M_1 accepts, accept and done.
   (d) $T \leftarrow 2T$

Each time we simulate both M_0 and M_1 for T steps. Because either M_0 rejects L or M_1 accepts L, our algorithm will terminate. So L is in COMP. □

Corollary. $HALT \notin$ co-R.E..

NOT HALT, the complement class of *HALT*, is in co-R.E., because $HALT \in$ R.E..

1.1 Remarks

R.E. is analogous to a $\exists$ quantifier: $L \in$ R.E. means for some $L' \in$ COMP,

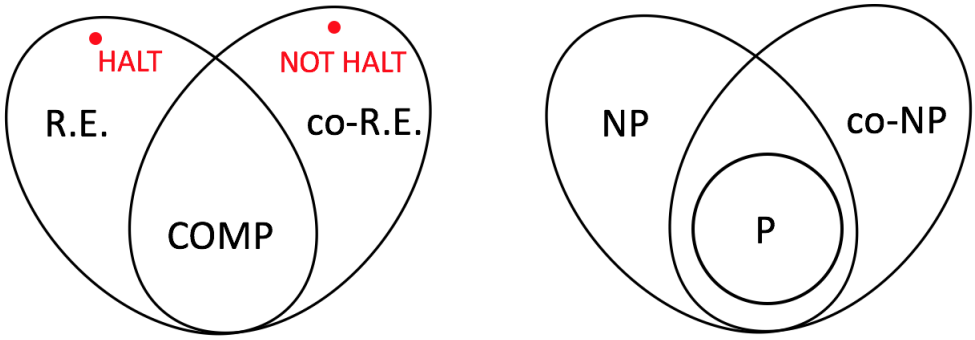
$$x \in L \Leftrightarrow, \exists T \ (x, T) \in L'.$$

co-R.E. is analogous to a $\forall$ quantifier: $L \in$ co-R.E. means for some $L' \in$ COMP,

$$x \in L \Leftrightarrow \forall T \ (x, T) \in L'.$$

The relation between NP and co-NP is similar. NP is the class of languages that accept strings having some $\exists$ quantified properties, while strings accepted by a co-NP language have some $\forall$ quantified properties. These classes will be introduced in the future lectures.

$P \subseteq \text{NP} \cap \text{co-NP}$. It is open whether $\text{P} = \text{NP}$(equivalently $\text{P} = \text{co-NP}$), $\text{NP} = \text{co-NP}$ or $\text{P} = \text{NP} \cap \text{co-NP}$.



2 Turing Reductions

Recall that $HALT = \{(M,x) \mid M \text{ halts on } x\}$. We define $ACCEPT = \{(M,x) \mid M \text{ eventually accepts } x\}$. We show that if $HALT \in \text{COMP}$, then $ACCEPT \in \text{COMP}$. If we can decide if $(M,x) \in HALT$, then we can decide if $(M,x) \in ACCEPT$ in the following way.

1. If $(M,x) \in HALT$, then
   (a) run M on x.
   (b) If M accepts x, then accept, otherwise reject.
2. Else reject.

What we have done is a Turing reduction from *ACCEPT* to *HALT*. We say there is a *Turing reduction* from language L to language L', denoted by $L \leq_T L'$, if there is an algorithm with a "sub-procedure" for L that recognizes L.

Example: $\overline{L} \leq_T L$ (by simply negating the answer from the oracle machine deciding L.)

Lemma 2.1.

1. If $L \leq_T L'$, and $L' \in \text{COMP}$, then $L \in \text{COMP}$.
2. $L \leq_T L'$, and $L' \leq_T L''$, then $L \leq_T L''$.

3 Mapping Reductions

We say there is a *mapping reduction* (also called *many-one reduction*) from language L to language L', denoted by $L \leq_m L'$ if there is a computable function f so that $x \in L$ iff $f(x) \in L'$.

A mapping reduction from *ACCEPT* to *HALT* is constructed as follows:

To decide whether $(M, x) \in ACCEPT$, we modify M to get a machine M' so that

1. M' simulates M, but
2. whenever M halts and accepts, M halts,
3. whenever M halts and rejects, M' loops forever.

Now we have $(M', x) \in HALT$ iff $(M, x) \in ACCEPT$.

Lemma 3.1.

1. If $L \leq_m L'$, and $L' \in \mathsf{COMP}$, then $L \in \mathsf{COMP}$.
2. $L \leq_m L'$, and $L' \leq_m L''$, then $L \leq_m L''$.
3. $L \leq_m L'$, and $L' \in \mathsf{R.E.}$, then $L \in \mathsf{R.E.}$.

Unlike Turing reductions, $\overline{L} \leq_m L$ might not hold true. Because in mapping reductions, we cannot use negation of the answer by the oracle machine of L, like we did in Turing reductions.

Define *NOT HALT* to be the complement of *HALT*. Does *NOT HALT* $\leq_m$ *HALT*? The answer is No, because *NOT HALT* $\notin \mathsf{R.E.}$, but *HALT* $\in \mathsf{R.E.}$, which contradicts the third point of Lemma 3.1.

HALT is R.E.-complete under mapping reductions. That means $\forall L \in \mathsf{R.E.}$, $L \leq_m$ *HALT*.

$\forall L \in \mathsf{R.E.}$, $L \leq_m ACCEPT$. Because $x \in L$ iff a Turing machine M_0 halts and accepts x, iff $(M_0, x) \in ACCEPT$. From $ACCEPT \leq_m HALT$, we get $L \leq_m HALT$.

3.1 Remarks

In general, given $L \leq L'$, ($\leq$ can be either Turing or mapping reductions)

1. If we know L' is easy, then we know L is also easy.
2. If we know L' is hard, then we know nothing about L.
3. If we know L is hard, then we know L' is also hard.
4. If we know L is easy, then we know nothing about L'.