Tutorial 5

Gidon Rosalki

2025-04-29

1 Computability classes R, RE

1.1 Definitions

Definition 1.1 (Language of a TM)**.** *The language of a Turing Machine M is*

$$L(M) = \{w \in \Sigma^* : M \text{ accepts } w\}$$

Definition 1.2 (Recognises)**.** *A Turing Machine M recognises L if* $L(M) = L$

Definition 1.3 (Decides)**.** *A Turing Machine M decides a language L if* $L(M) = L$*, and M halts on every input.*

Notice that if M decides L, M halts and rejects every $w \notin L$, but if M only recognizes L, M may also not halt on such words.

Definition 1.4 (RE)**.** *RE is the set of all recognizable languages, that is:*

$$RE = \{L : \text{There is a TM that recognizes } L\}$$

Definition 1.5 (R)**.** *R is the set of decidable languages, that is:*

$$R = \{L : \text{There is a TM that decides } L\}$$

1.2 Time bound acceptance is decidable

We can use the UTM to recognise acceptance of a word by a TM. We simulate the run of M on w, and accept only if M accepts. Therefore, $A_{\text{TM}} = \{\langle M, w\rangle : M \text{ accepts } w\} \in RE$. However, we will see later that A_{TM} is not decidable. However we shall show that a similar language that bounds the length of the run is decidable.

Theorem 1.

$$L = \{\langle M, w, k\rangle : M \text{ accepts } w \text{ within at most } k \text{ steps}\}$$

Proof . We will construct a TM T, that decides L. If will operate as follows on $\langle M, w, k\rangle$:

1. Simulate the run of M on w, for at most k steps, by using another TM on the side that counts the number of steps.
2. If M accepts w after at most k steps, T accepts, otherwise T rejects.

The correctness is as follows:

- By the definition of T, it accepts **if and only if** the run of M on w finishes within k steps, so it recognises L
- T halts on all inputs, since it simulates the run for a finite number of steps (k), and stops afterwards, so therefore it does not enter an infinite loop, and therefore decides L.

□

1.3 NON-EMPTY TM is recognisable

We will show that the language of encodings of TMs with a non empty language is recognisable (and will later show that it is not decidable). (As in, this is a TM that recognises machines that do not have an empty language).

Theorem 2.

$$NON-EMPTY_{TM} = \{\langle M\rangle : L(M) \neq \emptyset\} \in RE$$

Proof . We will create a TM T as follows:
For every $n \geq 0$:
For every $w \in \Sigma^* : |w| \leq n$:
Run M on w for at most n steps, and if M accepts w, then accept.

Correctness: We will prove this through two way containment. If $\langle M\rangle \in L(T)$, then by the construction, there exists $w \in \Sigma^*$ such that M accepts, so $\langle M\rangle \in NON-EMPTY_{TM}$.
If $\langle M\rangle \in NON-EMPTY_{TM}$, then there exists a word w that M accepts. We will denote by $k \in \mathbb{N}$ the length of the run of M on w. By the construction, the $i = \max\{|w|, k\}$th iteration, T will simulate the run of M on w, (because $|w| \leq i$), for at least k steps (since $k \leq i$), and hence M will reach an accepting state, and so T will accept. □

1.4 Closure properties of R, and RE

1.4.1 R

Theorem 3.

$$L \in R \implies \overline{L} \in R \ \left(L \cup \overline{L} = \Sigma^*\right)$$

Proof . Construction: Since $L \in R$, there exists a TM M that decides L. We will use M to construct a new machine T, that decides $\overline{L}$. This is done by swapping the two states q_{acc} and q_{rej}.

Correctness: First we will note that since M stops on all inputs, the machine T also stops on all inputs, as their only differences is the name of the final states. $L(T) = \overline{L}$, since

$$T \text{ accepts } w \Leftrightarrow M \text{ rejects } w \Leftrightarrow w \notin L \Leftrightarrow w \in \overline{L}$$

□

Theorem 4 (R is closed to union)**.**

$$L_1, L_2 \in R \implies L_1 \cup L_2 \in R$$

Proof . Construction: We will create a TM T that operates as follows: Given $w \in \Sigma^*$:

1. We will run M_1 that decides L_1 on w, and if it accepts, T will also accept
2. We will run M_2 that decides L_2 on w, and if it accepts, T will also accept
3. Otherwise, T rejects.

Remark: To accomplish this, T can duplicate the word w to a more distant location on the tape and mark it with a special symbol. Whenever M_1's computation reaches this symbol, T uses a procedure to move w even further along the tape. Upon M_1's termination, if acceptance does not occur, T clears the tape used by M_1, and repositions the head to the beginning of the duplicated copy of w and initiates the computation of M_2.

Correctness: Since M_1 and M_2 halt on all inputs, so does T. From the construction of T, it is clear that T accepts a word **if and only if** M_1 accepts the word, or M_2 accepts the word, and so $L(T) = L_1 \cup L_2$ □

Theorem 5 (R is closed to intersection)**.**

$$L_1, L_2 \in R \implies L_1 \cap L_2 \in R$$

Proof . We can show it by a constructing a Turing machine that operates like the machine used in Theorem 4, with the difference that it accepts **if and only if** both machines accept.

Remark: It also follows from closure, to complement and union since: $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ □

1.4.2 RE

When dealing with the closure properties of RE, we are dealing with TMs that may not halt.

Theorem 6.

$$L_1, L_2 \in R \implies L_1 \cap L_2 \in R$$

Proof . We will use the same construct as we used as we used to show intersection over R, since if one of the machines does not halt, then T does not halt, which is the desired behaviour. The word is then not accepted by the non halting machine, and hence is not in the intersection. □

Theorem 7.

$$L_1, L_2 \in R \implies L_1 \cup L_2 \in R$$

Proof . We may not simply use the same construct, since if M_1 does not halt, but M_2 would have halted, then M_1 will make it appear as though the word is rejected by T. We shall instead attempt to run both machines "in parallel".

Construction: Let there be M_1, M_2 machines that recognise L_1, L_2 respectively. We will define T that recognises $L_1 \cup L_2$ as follows:

1. For every $n \geq 0$:
   (a) Run M_1 on w for n steps, and if M_1 accepts, then accept
   (b) Run M_2 on w for n steps, and if M_2 accepts, then accept

Correctness: □

2 Reductions

Definition 2.1 (Output of a TM). *If a TM M halts on an input w, then we will define $M(w)$ to be what is left on the tape at the end of the run (neglecting spaces).*

Definition 2.2 (Computable). *A function $f : \Sigma^* \to \Sigma^*$ is called **computable** if there exists an TM M_f such that for every $w \in \Sigma^*$, it holds that $M_f(w) = f(w)$*

Definition 2.3 (Reduction). *A **reduction** from a language $A \subseteq \Sigma^*$ to the language $B \subseteq \Sigma^*$ is a computable function $f : \Sigma^* \to \Sigma^*$, such that for all $x \in \Sigma^*$,*

$$x \in A \Leftrightarrow f(x) \in B$$

*If such a function exists, then we denote $a \leq_m B$. A TM that computes f is called a **reduction machine**.*

2.1 Example: Parity languages

Consider the following languages over $\Sigma = \{a, b\}$:

$$L_1 = \{w \in \Sigma^* : 2 \mid |w|\} \text{ (Even length strings) } L_2 = \{w \in \Sigma^* : 2 \nmid |w|\} \text{ (Odd length strings)}$$

We shall claim that $L_1 \leq_m L_2$:

2.1.1 Reduction 1: solve the problem

We can define $f : \Sigma^* \to \Sigma^*$ as follows:

$$f(w) = \begin{cases} a, & \text{if } |w| \mod 2 = 0 \\ aa, & \text{if } |w| \mod 2 = 1 \end{cases}$$

f is a computable function, so we can determine the length of a word using a TM, and determine if it is even or odd. We shall now prove that $x \in L_1 \Leftrightarrow f(x) \in L_2$ holds:

2.1.2 Reduction 2: transfer the problem

2.2 NON-EMPTY TM

Theorem 8.

$$HALT_{TM} \leq_m NON - EMPTY_{TM}$$

Proof . How to approach these reduction problems? Ask yourself the following:

1. **What should we construct?** That one is easy - it is always a computable function: $f : \Sigma^* \to \Sigma^*$

2. **What is the type of the input and output of f?** Naturally, the input and the output can be any word, but since handling invalid inputs is usually an easy task, we will focus on "valid" inputs. In our case, f takes an encoding of a TM, and a word, $\langle M, w\rangle$, as input and outputs an encoding of a TM $\langle T_{M,w}\rangle$

3. **What condition should f satisfy?** By the definition of reduction and the definitions of the languages, M halts on w **if and only if** the language of T is not empty.

Once we understand what to construct, we focus on how to construct such a function:
It is tempting here to use a function

$$f(\langle M, w\rangle) = \begin{cases} \langle M_{ALL}\rangle, & \text{if } M \text{ halts on } w \\ \langle M_{EMPTY}\rangle, & \text{otherwise} \end{cases}$$

but this is impossible, thanks to the proof of the halting problem, so don't do that!
To show that f is a reduction, we should show that it is computable, namely that there exists a TM M_f that computes f. In our example, M_f cannot check if M halts on w, so we should probably encode M and w inside $T_{M,w} \overset{def}{=} f(\langle M, w\rangle)$, so as part of its run on an input x, it can simulate the run of M on w. The intuition is that if M does not halt on w, then T should not accept, no matter what is its input x. If M does halt on w, then T should accept some input, and for simplicity we can design it to accept every input x.

Construction: Let $f : \Sigma^* \to \Sigma^*$ be the function that returns ε for an "invalid" input (that is, an input that is not an encoding of a TM, and a word), and for a valid input returns

$$f(\langle M, w\rangle) = \langle T_{M,w}\rangle$$

where $T_{M,w}$ operates as follows on an input x:

1. Delete x from the tape

2. Write w on the tape
3. Run M on w as a procedure
4. If M reaches a final state, (may it be accept, or reject), then accept

Computability: f is computable for the following reasons:

1. Checking the validity of the input (whether or not it is an encoding of a TM, and a word) is computable
2. Given a TM M and a word w constructing such a TM is computable by "combining" a TM that deletes its input, a TM that writes w on its tape, and M.

Correctness: For invalid invalid input (that which is not in $HALT_{TM}$), the reduction returns an invalid input, (that which is not in $NON-EMPTY_{TM}$). For a valid input:

1. If

□

2.3 Reduction theorem

Theorem 9 (Reduction theorem). *Let $L_1, L_2 \subseteq \Sigma^*$, such that $L_1 \leq_m L_2$. Then $L_2 \in R \implies L_1 \in R$, and equivalently $L_1 \notin R \implies L_2 \notin R$. Similarly, $L_2 \in RE \implies L_1 \in RE$, and equivalently $L_1 \notin RE \implies L_2 \notin RE$.*

Intuition: Since $L_1 \leq_m L_2$, then any task in L_2 is at least as difficult to carry out as the task in L_1. If there is something that we know how to do with L_2, be that deciding, recognising, or recognising its complement, then we can do the same for L_1 by first applying the reduction to L_2. Equivalently, if there is something we know to be impossible with L_1, then the same applies to L_2.

Example: By Theorem 8, we saw that $HALT_{TM} \leq_m NON-EMPTY_{TM}$, and we saw in the lecture that $HALT_{TM} \notin R$, so by the reduction theorem, it also holds that $NON-EMPTY_{TM} \notin R$.