CS21004 Formal Languages and Automata Theory, Spring 2019–2020

Reductions and Undecidability

---

1. Prove that the language

$$\text{WB} = \{M \# w \mid M \text{ writes the blank symbol in some step on input } w\}$$

is not recursive.

Solution We make a reduction $\text{HP} \leqslant_m \text{WB}$. The input is an instance $M \# w$ of HP. The output is a Turing machine N and an input v for N such that N writes its blank symbol on its tape in some step while working on v, if and only if M halts on w. The tape of N is composed of two tracks. The upper track is used to simulate M on w, whereas the lower track controls the writing of the blank symbol. More precisely, the tape alphabet of N is $\Gamma_N = \Gamma_M \times \{\square, \blacksquare\}$, where $\square$ is the blank symbol of M. The blank symbol for N is $(\square, \square)$ (blank on both the tracks). We also take $Q_N = Q_M \cup \{t', r'\}$, where t' and r' are the new accept and reject states of N.

The input w for M is copied to the upper track, and the lower track is kept blank. During the simulation of M on the upper track, N always writes $\blacksquare$ on the lower track. That is, for each $\delta_M(p,a) = (q,b,d)$, N has two transitions: $\delta_N(p,(a,\square)) = (q,(b,\blacksquare),d)$ and $\delta_N(p,(a,\blacksquare)) = (q,(b,\blacksquare),d)$. If the simulation of M on w halts, N writes $(\square,\square)$ at the current head position (if the head is at the leftmost cell, then the head moves one cell right before writing the blank), and itself halts (or loops or does whatever it likes). For example, N may have the new transitions $\delta_N(t,(\triangleright,\triangleright)) = (t,(\triangleright,\triangleright),R)$, $\delta_N(r,(\triangleright,\triangleright)) = (r,(\triangleright,\triangleright),R)$, and $\delta_N(t,(*,*)) = (t',(\square,\square),R)$ and $\delta_N(r,(*,*)) = (r',(\square,\square),R)$ for any other tape symbol $(*,*)$.

2. **(a)** Prove that the language

$$E_{2020} = \{M \mid M \text{ halts on exactly 2020 inputs}\}$$

is not r.e.

Solution We use a reduction $\overline{\text{HP}} \leqslant_m E_{2020}$. The input is an instance $M \# w$ for $\overline{\text{HP}}$, and the output is a Turing machine N that halts on exactly 2020 inputs if and only if M does not halt on w.

Fix any of 2020 strings $v_1, v_2, v_3, \ldots, v_{2020}$. For example, if $a \in \Sigma_M$, you can take $v_i = a^i$ for $i = 1,2,3,\ldots,2020$. The machine N on input $v = w$ proceeds as follows.

1. Check whether $v = v_i$ for some $i = 1,2,3,\ldots,2020$. If so, accept, and halt.
2. Simulate M on v (same as w).
3. If the simulation halts, accept and halt.

If M does not halt on w, then N accepts only the strings $v_1, v_2, v_3, \ldots, v_{2020}$. If M halts on w, then N accepts all the strings. This establishes the correctness of the reduction.

(b) Prove that the language

$$AL_{2020} = \{M \mid M \text{ halts on at least 2020 inputs}\}$$

is r.e. but not recursive.

Solution A Turing machine K can simulate M on all possible strings on a time-sharing basis. If M accepts at least 2020 inputs, K eventually detects it, and accepts and halts. If M halts on less than 2020 inputs, the simulation of K never halts, so this is a case of looping and implicit rejection.

In order to prove non-recursiveness, we use a reduction $\text{HP} \leqslant_m AL_{2020}$. The input is an instance $M \# w$ for HP, and the output is a TM N that halts on at least 2020 strings if and only if M halts on w. The input for N is v. N first simulates M on w. If the simulation halts, N accepts v and halts. In this case, N accepts all $v \in \Sigma_N^*$. In particular, N halts on at least 2020 inputs. However, if the simulation of M on w does not halt, N has no chance to accept and halt on any input v, so in this case, N halts on $0 < 2020$ inputs.

3. Let $nsteps(M,w)$ denote the number of steps of M on w. If M loops on w, take $nsteps(M,w) = \infty$. If N also loops on v, take $nsteps(M,w) = nsteps(N,v)$. Recursive / r.e. but not recursive / non-r.e.? Prove.

(a) $L_a = \{M \# N \mid nsteps(M,\varepsilon) < nsteps(N,\varepsilon)\}$.

Solution *R.E. but not recursive* A two-tape TM K can simulate M on ε on one tape, and N on ε on the other tape in a round-robin fashion. If the simulation of M halts before that of N, K accepts. If the simulation of N halts before

that of M, K rejects. If the two simulations halt after the same number of steps, K rejects. If neither simulation halts, K continues with the simulation (that is, loops) for ever, and never accepts. This shows that L_a is r.e.

In order to show that L_a is not recursive, we propose a reduction HP $\leqslant_m L_a$. Upon input $M \# w$ (an instance of HP), the reduction algorithm outputs $N_1 \# N_2$ (an instance of L_a) such that N_1 halts in fewer steps than N_2 upon input ε if and only if M halts on w.

N_1 upon input v_1 first checks whether $v_1 = \varepsilon$. If not, N_1 enters an infinite loop. If $v_1 = \varepsilon$, N_1 simulates M on w, and accepts if M halts.

Upon any input v_2 , the other machine N_2 enters an infinite loop.

If M halts on w, N_1 halts on ε (that is, takes a finite number of steps before halting), whereas N_2 loops on ε (infinite steps), that is, $nsteps(N_1, \varepsilon) < nsteps(N_2, \varepsilon)$. On the other hand, if M does not halt on w, both N_1 and N_2 loop on input ε, that is, $nsteps(N_1, \varepsilon) = nsteps(N_2, \varepsilon) = \infty$.

(b) $L_b = \{M \# N \mid nsteps(M, \varepsilon) \leqslant nsteps(N, \varepsilon)\}$.

Solution *Not r.e.* The language $\overline{L_a} = \{M \# N \mid nsteps(M, \varepsilon) \geqslant nsteps(N, \varepsilon)\}$ is not r.e., since if both L_a and $\overline{L_a}$ are r.e., L_a is recursive. The simple reduction $\overline{L_a} \leqslant_m L_b$ converting $M \# N$ to $N \# M$ shows that L_b is not r.e.

(c) $L_c = \{M \# N \mid nsteps(M, w) < nsteps(N, v)$ for some $w, v\}$.

Solution *R.E. but not recursive.* An NTM K can non-deterministically guess w and v for which $nsteps(M, w) < nsteps(N, v)$, and verify it by simulating M on w and N on v in a round-robin fashion on two tapes. If the verification succeeds, K accepts and halts. If there do not exist any w and v for which $nsteps(M, w) < nsteps(N, v)$, then all guesses of K fail to accept.

In order to prove non-recursiveness, we use a reduction HP $\leqslant_m L_c$. Given an input $M \# w$ (an instance of HP), the reduction algorithm generates a pair of machines N_1, N_2 such that $nsteps(N_1, v_1) < nsteps(N_2, v_2)$ for some v_1, v_2 if and only if M halts on w. Without loss of generality, we can take $v_1 = v_2 = \varepsilon$.

N_1 on input v_1 does the following.

1. Check if $v_1 = \varepsilon$. If not, accept, and halt.
2. If $v_1 = \varepsilon$, simulate M on w.
3. If the simulation halts, accept and halt.

N_2 on input v_2 does the following.

1. Check if $v_1 = \varepsilon$. If not, accept, and halt.
2. If $v_1 = \varepsilon$, enter an infinite loop.

If M halts on w, $nsteps(N_1, \varepsilon) < \infty$. If M does not halt on w, then $nsteps(N_1, \varepsilon) = \infty$. In both the cases, we have $nsteps(N_2, \varepsilon) = \infty$.

(d) $L_d = \{M \# N \mid nsteps(M, w) < nsteps(N, v)$ for all $w, v\}$.

Solution *Not recursively enumerable.* We propose a reduction $\overline{\text{HP}} \leqslant_m L_d$. On input $M \# w$ (an instance for $\overline{\text{HP}}$), the reduction algorithm produces two TMs N_1 and N_2 such that $nsteps(N_1, v_1) < nsteps(N_2, v_2)$ for all v_1, v_2 if and only if M does not halt on w.

N_1 on input v_1 does the following.

1. Simulate M on w for $|v_1|$ steps.
2. If the simulation halts within these many steps, enter an infinite loop.
3. If the simulation does not halt within these many steps, accept v_1 and halt.

N_2 on input v_2 does the following.

1. Simulate M on w.
2. If the simulation halts, accept v_2 and halt.

If M does not halt on w, M does not halt in any finite number of steps. So N_1 halts in Step 3 irrespective of its input. In particular, $nsteps(N_1, v_1)$ is finite for all v_1. On the other hand, the simulation of M on w by N_2 keeps running forever, so $nsteps(N_2, v_2) = \infty$ for any input v_2.

If M halts on w, say, in n steps, N_2 gets a chance to come to Step 2 and halt. So $nsteps(N_2, v_2)$ is finite for all v_2 in this case. On the other hand, for any input v_1 with $|v_1| \geqslant n$, the limited-time simulation by N_1 detects

the halting of M on w. In this case, N_1 goes to Step 2, and enters an infinite loop, that is, $nsteps(N_1, v_1) = \infty$ whenever $|v_1| \geqslant n$.

4. Prove that the following languages are not recursive.

(a) $\{M \# N \mid \mathscr{L}(M) = \mathscr{L}(N)\}$.

Solution *Reduction from* HP

Input: $M \# w$.

Output: $N_1 \# N_2$ such that $\mathscr{L}(N_1) = \mathscr{L}(N_2)$ if and only if M halts on w.

N_1 on input v_1 accepts and halts.

N_2 on input v_2 simulates M on w. If the simulation halts, N_2 accepts v_2 and halts.

If M halts on w, $\mathscr{L}(N_1) = \mathscr{L}(N_2) = \Sigma^*$. If M does not halt on w, $\mathscr{L}(N_2) = \emptyset$, whereas $\mathscr{L}(N_1) = \Sigma^*$ as in the other case.

(b) $\{M \# N \mid \mathscr{L}(M) \subseteq \mathscr{L}(N)\}$.

Solution The reduction of Part (a) works for this part too.

(c) $\{M \# N \mid \mathscr{L}(M) \cap \mathscr{L}(N) = \emptyset\}$.

Solution *Reduction from* $\overline{HP}$ [This proves even non-r.e.-ness, but that is perfectly OK]

Input: $M \# w$.

Output: $N_1 \# N_2$ such that $\mathscr{L}(N_1) \cap \mathscr{L}(N_2) = \emptyset$ if and only if M does not halt on w.

N_1 on input v_1 accepts v_1 if and only if $|v_1|$ is even.

N_2 on input v_2 simulates M on w for $|v_2|$ steps. If the simulation halts in these many steps, accept v_2. If the simulation does not halt in these many steps, accept v_2 if and only if $|v_2|$ is odd.

We have $\mathscr{L}(N_1) = \{v \in \Sigma^* \mid |v| \text{ is even}\}$ irrespective of whether M halts on w or not.

On the other hand, if M does not halt on w, then $\mathscr{L}(N_2) = \{v \in \Sigma^* \mid |v| \text{ is odd}\}$, whereas if M halts on w in n steps, $\mathscr{L}(N_2)$ consists of all strings of odd lengths $< n$, and all strings of both odd and even lengths $\geqslant n$.

(d) $\{M \# N \# P \mid \mathscr{L}(M) \cap \mathscr{L}(N) = \mathscr{L}(P)\}$.

Solution *Reduction from* HP

Input: $M \# w$.

Output: $N_1 \# N_2 \# N_3$ such that $\mathscr{L}(N_1) \cap \mathscr{L}(N_2) = \mathscr{L}(N_3)$ if and only if M halts on w.

N_1 on any input v_1 accepts v_1, so $\mathscr{L}(N_1) = \Sigma^*$.

N_2 on any input v_2 accepts v_2, so $\mathscr{L}(N_2) = \Sigma^*$ too.

N_3 on input v_3 first simulates M on w. If the simulation halts, N_3 accepts v_3. We therefore have

$$\mathscr{L}(N_3) = \begin{cases} \Sigma^* & \text{if } M \text{ halts on } w, \\ \emptyset & \text{if } M \text{ does not halt on } w. \end{cases}$$

5. Prove that neither the language $\text{REG} = \{M \mid \mathscr{L}(M) \text{ is regular}\}$ nor its complement is r.e.

Solution **REG is not r.e.**

Use reduction from $\overline{\text{HP}}$. The input to the reduction algorithm is $M \# w$, and its output is a TM N such that $\mathscr{L}(N)$ is regular if and only if M does not halt on w. N, on input v, does the following.

1. Simulate M on w.
2. If the simulation halts, N accepts v if and only if $v = 0^p$ for some prime p.

We have

$$\mathscr{L}(N) = \begin{cases} \emptyset & \text{if } M \text{ does not halt on } w, \\ \{0^p \mid p \text{ is a prime}\} & \text{if } M \text{ halts on } w. \end{cases}$$

$\overline{\text{REG}}$ is not r.e.

Use reduction from $\overline{\text{HP}}$. The input to the reduction algorithm is $M \# w$, and its output is a TM N such that $\mathscr{L}(N)$ is not regular if and only if M does not halt on w. N, on input v, does the following.

1. Simulate M on w for $|v|$ steps.
2. If the simulation halts in these many steps, reject v.
3. If the simulation does not halt in these many steps, accept v if and only if $v = 0^p$ for some prime p.

We have

$$\mathscr{L}(N) = \begin{cases} \{0^p \mid p \text{ is a prime}\} & \text{if } M \text{ does not halt on } w, \\ \{0^p \mid p \text{ is a prime} < n\} & \text{if } M \text{ halts on } w \text{ in } n \text{ steps.} \end{cases}$$

The second set is finite (and so regular).

6. Prove that the following problems on a TM M are decidable.

(a) Decide whether M halts on some input within 2020 steps.

Solution This is the complement of the problem whether M takes more that 2020 steps on all inputs.

(b) Decide whether M on a given input w moves left at least ten times.

Solution Simulate M on w for $|w| + |Q| + 10$ steps. If the simulation makes ten (or more) left movements, accept. Otherwise, reject. The correctness of the rejection decision comes from the fact that if M makes less than ten left movements in the above number of iterations, the head is scanning only the blank symbol. Indeed within $|w| + 10$ steps, the head first starts scanning the blank symbol. If $|Q|$ more steps are allowed, then some state must be repeated indicating that the machine has entered an infinite loop.

7. Is the problem whether a Turing machine on a given input reenters the start state decidable or not? Prove.

Solution *Undecidable*. Use reduction from HP. Given an input $M \# w$ (an instance for HP), we create a machine M' which reenters its start state if and only if M halts on w. The following modifications are done on M to get M'.

1. Mark the start state s of M as a non-start state.
2. Add a new start state s'.
3. Mark the accept state t and the reject state r of M as non-halting.
4. Add a new accept state t' and a new reject state r'. We have $Q' = Q \cup \{s', t', r'\}$.
5. Add a new symbol ■ to the tape alphabet of M. Denote $\Gamma' = \Gamma \cup \{\blacksquare\}$.
6. Add the following transitions:

$$\begin{aligned} \delta(s', \triangleright) &= (s', \triangleright, R) \\ \delta(s', a) &= (s, a, L) \text{ for any } a \in \Gamma \setminus \{\triangleright\} \\ \delta(t, \triangleright) &= (t, \triangleright, R) \\ \delta(r, \triangleright) &= (r, \triangleright, R) \\ \delta(t, a) &= (s', \blacksquare, R) \text{ for any } a \in \Gamma \setminus \{\triangleright\} \\ \delta(r, a) &= (s', \blacksquare, R) \text{ for any } a \in \Gamma \setminus \{\triangleright\} \\ \delta(s', \blacksquare) &= (t', \blacksquare, L) \\ \delta(p, \blacksquare) &= (r', \blacksquare, L) \text{ for any } p \neq s', t', r'. \end{aligned}$$

M', on input w, starts in the new start state s'. After two moves, it switches to the old start state (the first two added transitions). After that, the old transitions of M take effect, and M' perfectly simulates M on w. If the simulation halts in state t or r, M' does a little bit of work before halting. It writes the new tape symbol ■ in the current head position (skipping the leftmost cell if necessary), and comes back to this cell, and accepts by going to the new accept state t'. The last transition added above should never be executed, but given to complete the specification of the machine M'. If M does not halt on w, M' never reaches t or r, so s' is never reentered by M'.

Remarks: In Q1 and Q7, the reduction algorithms are specified explicitly. Do this whenever possible. In general, the reduction algorithm is cumbersome. So it suffices to describe the behavior of the output. We leave it to our intuition that the reduction algorithm (a total TM) can make this conversion. If the input to the reduction algorithm is a machine M and its input w, the reduction algorithm can embed the information of M and w in the finite control (transition function) of the output machine. So the output machine can write w to a tape, and simulate M on w.