Homework II, Theory of Computation 2025

Submission: **The deadline for Homework 2 is 23:59 on 17 April.** Please submit your solutions on Moodle. Typing your solutions using a typesetting system such as LaTeX is strongly encouraged! If you must handwrite your solutions, write cleanly and with a pen. Messy and unreadable homeworks will not be graded. No late homeworks will be accepted.

Writing: Please be precise, concise and (reasonably) formal. Keep in mind that many of the problems ask you to provide a proof of a statement (as opposed to, say, just to provide an example). Therefore, make sure that your reasoning is correct and there are no holes in it. A solution that is hard/impossible to decipher/follow might not get full credit (even if it is in principle correct). You do not need to reprove anything that was shown in the class—just state clearly what was proved and where.

Collaboration: These problem sets are meant to be worked on in groups of 2–4 students. Please submit only one writeup per team—it should contain the names of all the students. You are strongly encouraged to solve these problems by yourself. If you must, you may use books or online resources to help solve homework problems, but you must credit all such sources in your writeup and you must never copy material verbatim. Even though only one writeup is submitted, it is expected that each one of the team members is able to fully explain the solutions if requested to do so.

Grading: Each of the two problems will be graded on a scale from 0 to 5.

Warning: Your attention is drawn to the EPFL policy on academic dishonesty. In particular, you should be aware that copying solutions, in whole or in part, from other students in the class or any other source (e.g., ChatGPT) without acknowledgement constitutes cheating. Any student found to be cheating risks automatically failing the class and being referred to the appropriate office.

Homework 2

1 Classify each of the following languages into one of the following three categories:

decidable, undecidable but recognizable, unrecognizable

Justify your answers with proofs. (Recall below that $L(M) = \{w : M \text{ accepts } w\}$ is the language recognized by M.)

1a $L_1 = \big\{\langle M\rangle : M \text{ is a TM and } L(M) \text{ is decidable}\big\}$

1b $L_2 = \big\{\langle M, N\rangle : M \text{ and } N \text{ are TMs and } L(M) \cap L(N) \neq \emptyset\big\}$

1 Solution to Question 1

- L_1 is an **unrecognisable** language.

  Recall from the course that language $\overline{A_{TM}}$, defined as

  $$\overline{A_{TM}} \coloneqq \{\langle M, w\rangle : M \text{ does not accept } w\}$$

  is an unrecognisable language. If we show a reduction $\overline{A_{TM}} \leq_m L_1$ then we prove that L_1 is unrecognisable too which follows from the theorem shown in class:

  If $A \leq_m B$ and B is recognisable, then A is recognisable.

  We aim to build a function $f(\langle M, w\rangle) = \langle M'\rangle$ which maps a pair of a Turing machine and a string to a Turing machine so that M accepts w if and only if $L(M')$ is undecidable. We define M' so that on an input string y

  1. M' first simulates M on w.
  2. In case if this computation halts and accepts, M' reads string y. If y has the form $\langle N, z\rangle$ (that is, if the input string contains one special symbol ','), M' interprets N as a code for a Turing machine and z as the input and simulates N on z.
  3. If at any step the computation fails or rejects, M' enters an infinite loop.

  By Church-Turing thesis, a code for the described Turing machine M' can be obtained by executing a Turing machine algorithm on input $\langle M, w\rangle$, so f is computable. It remains to show that that M accepts w if and only if $L(M')$ is undecidable:

  1. If M accepts w then M only reaches an accept state if the input y has the form $\langle N, z\rangle$ and if N accepts z. In other words, $L(M') = A_{TM}$ which is known to be an undecidable language.
  2. If M rejects w or loops on w then M' enters an infinite loop and does not accept any input y. In other words, $L(M') = \emptyset$, which is a decidable language.

- L_2 is an **undecidable** but a **recognisable** language.

  We prove the two statements separately, starting with undecidability. From the course we know that A_{TM} is an undecidable language, so it suffices to show a reduction $A_{TM} \leq_m L_2$. In other words, we aim to build a function $f(\langle M, w\rangle) = \langle M', N'\rangle$ which maps a pair of a Turing machine and a string to two Turing machines so that M accepts w if and only if the languages of these two Turing machines have at least one string in common. We define $M' = N'$ so that on an input string y

1. M' first simulates M on w.
2. In case if this computation halts and accepts, M' accepts y.
3. Otherwise, M' enters an infinite loop.

By Church-Turing thesis, a code for the described Turing machine M' can be obtained by executing a Turing machine algorithm on input $\langle M, w\rangle$, so f is computable. It remains to show that that M accepts w if and only if $L(M') \cap L(N') \neq \emptyset$ or, equivalently, $L(M') \neq \emptyset$:

1. If M accepts w then M' accepts all strings which means that $L(M) = \Sigma^* \neq \emptyset$.
2. If M rejects w or loops on w then M' enters an infinite loop and does not accept any input y. In other words, $L(M') = \emptyset$.

To show that L_2 is recognisable, we provide a Turing machine algorithm which recognises L_2. We would want to check for every string in Σ^* whether M and N accept it – if such a string exists, we report that $\langle M, N\rangle \in L_2$, and if not then our algorithm will never halt and therefore $\langle M, N\rangle \notin L_2$. We cannot, however, simply check all the strings one by one since M and N might never halt on some strings. We have to carefully check all the strings in parallel, for example, with the algorithm below.

1. Enumerate all the strings in Σ^* in an arbitrary order (for example, start by enumerating all strings of length one, then length 2, and so on).
2. If at any step of the algorithm below we discover a string w_i such that both M and N accept w_i, then accept the pair $\langle M, N\rangle$:
3. Complete one step of simulating M on string w_1.
4. Complete one step of simulating M on strings w_1 and w_2.
5. . . .

These useful scheme of executing multiple computations in parallel is called *dovetailing*. By Church-Turing thesis, there exists a Turing machine which performs this algorithm.

Grading scheme:

- 2.5 points for correctly proving that L_1 is unrecognisable. 1 point for a proof with an incorrect reduction function f.
- 2.5 points for correctly proving that L_2 is undecidable but recognisable.

The score may be lowered because of alternative errors; these errors are outlined in the comment section of the submission.

2 **Busy Beaver.** Define the n-th *Busy Beaver* number, denoted $\mathrm{BB}(n)$, as the largest number $k \in \mathbb{N}$ such that there exists a TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\mathrm{halt}})$ such that

(1) M has $|Q| = n + 1$ states (that is, n states in addition to its halting state q_{halt}),
(2) M has a binary input/tape alphabet, $\Sigma = \Gamma = \{0, 1\}$,
(3) M, on the empty input ε, halts with 1^k written on its tape.

In other words, amongst all $(n+1)$-state TMs that halt on the empty input, what is the longest all-1 string that it can output? Note that $\mathrm{BB}(n)$ is always finite, as there are only finitely many distinct TMs satisfying (1)–(3) for any given n.

Show that the function $\mathrm{BB}\colon \mathbb{N} \to \mathbb{N}$ is not computable. That is, show that there is no TM that on input n (given in binary) will always halt with $\mathrm{BB}(n)$ (in binary) on its tape.

(*Hint: If* BB *were computable, how would you decide the Halting problem?*)

Solution: Assume for contradiction that BB is computable. We will show that this leads to a computable solution to the Halting Problem.

For a given Turing machine M and input $w \in \{0,1\}^\star$, consider the Turing machine $T_{\langle M,w\rangle}$, which does the following on empty input:

- **Run** M on w while recording somewhere on the tape the number of the current step.
- If $M(w)$ halts, use the recorded number of steps to write as many 1s on the tape, delete everything else, and halt.

Creating the description of $T_{\langle M,w\rangle}$ from $\langle M,w\rangle$ is clearly a computable operation. Now, we design the following Turing machine H, which will solve the Halting problem:

$H =$ "On input $\langle M,w\rangle$:

1. Create the description of $T_{\langle M,w\rangle}$. Calculate the number of states of $T_{\langle M,w\rangle}$: $n = |Q(T_{\langle M,w\rangle})|$.
2. Compute $B = \mathrm{BB}(n-1)$.
3. Run M on w.
4. If $M(w)$ halts in B steps, **accept**. Otherwise, **reject**.

Notice that H solves the Halting Problem because if $M(w)$ halts after more than $\mathrm{BB}(n-1)$ steps, then $T_{\langle M,w\rangle}$ would output more than $\mathrm{BB}(n-1)$ 1s and halt on the empty input – a contradiction.

Grading Scheme:

- 2 points for correctly defining $T_{\langle M,w\rangle}$. 0.5-point deduction for small errors in the description.
- 2 points for correctly defining H. 0.5-point deduction for small errors in the description.
- 1 point for correct argument that H solves the Halting Problem.
- 0.5-point deduction for poorly structured solution with ambiguous arguments.

Alternative solutions will be graded on a case-by-case basis.